CrowPanel Basic HMI ESP32 Display (5.0 / 7.0-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.

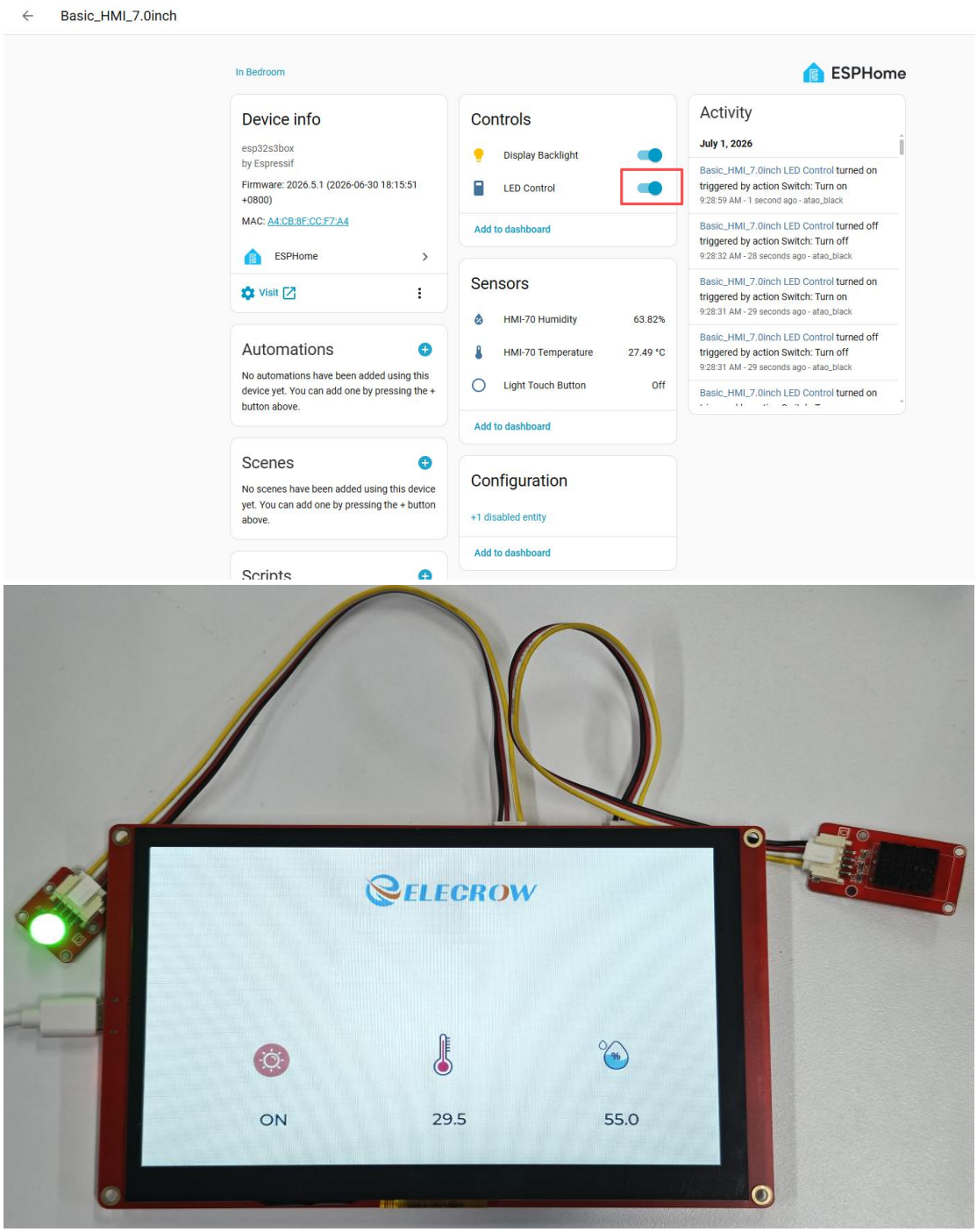
So let's move on to learn how to install the ESPHome environment, allowing you to edit code on it to achieve reading temperature and humidity data and remotely control the on and off of LEDs.

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.

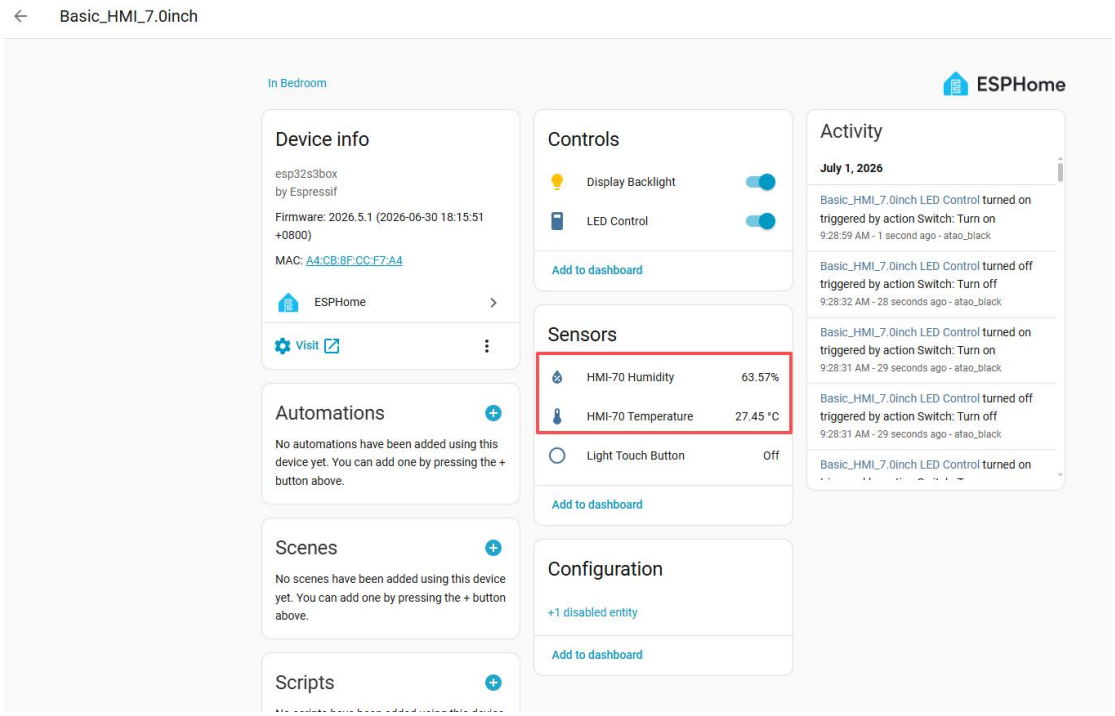
3.Project operation effect illustration

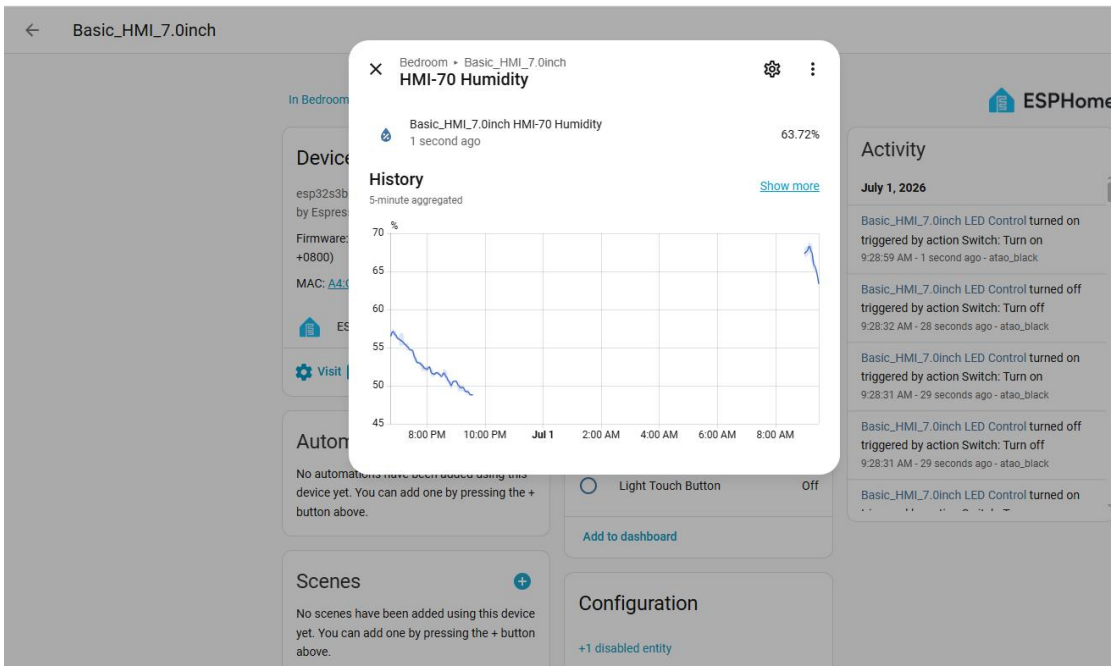
Next, when you click the switch of the LED, you will be able to see that the indicator light on the screen turns on, and the feedback from the LED is also quite obvious.

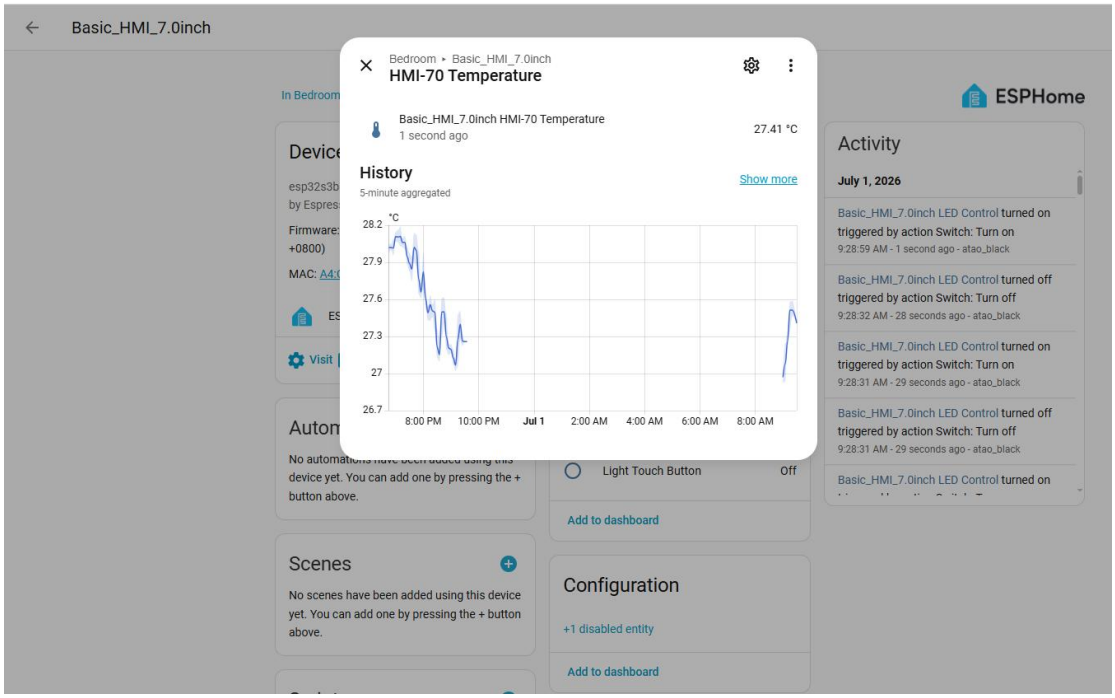


Next, when you click the switch of the LED, you will be able to see that the display light on the screen turns off, and the feedback from the LED is also quite obvious.

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







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.

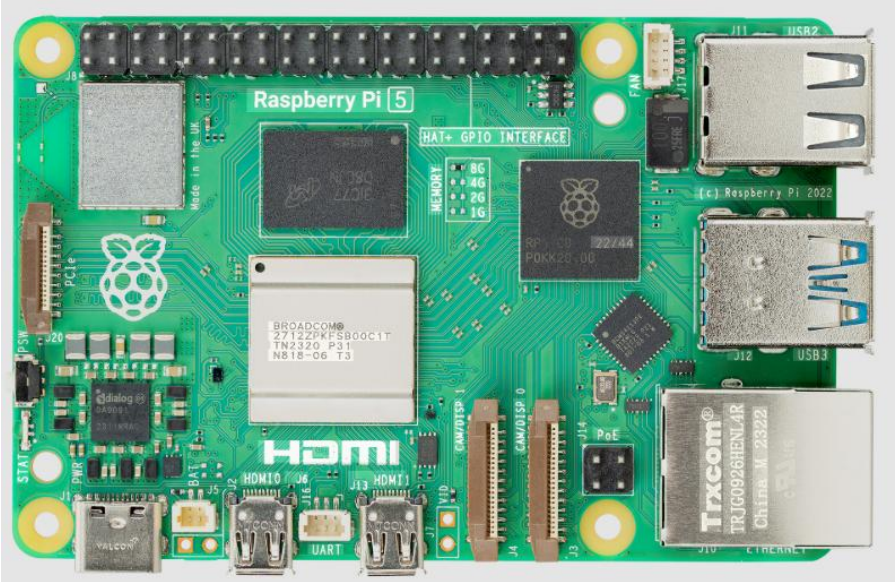
It works on computers, small servers, or even dedicated smart home hubs, and lets you manage every smart product you own (lights, thermostats, sensors, your rotary screen, etc.) in one unified interface—regardless of the brand or technology the device uses. For your rotary screen, HA acts as the "command 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 (5.0-inch \ 7.0-inch);
purchase link :

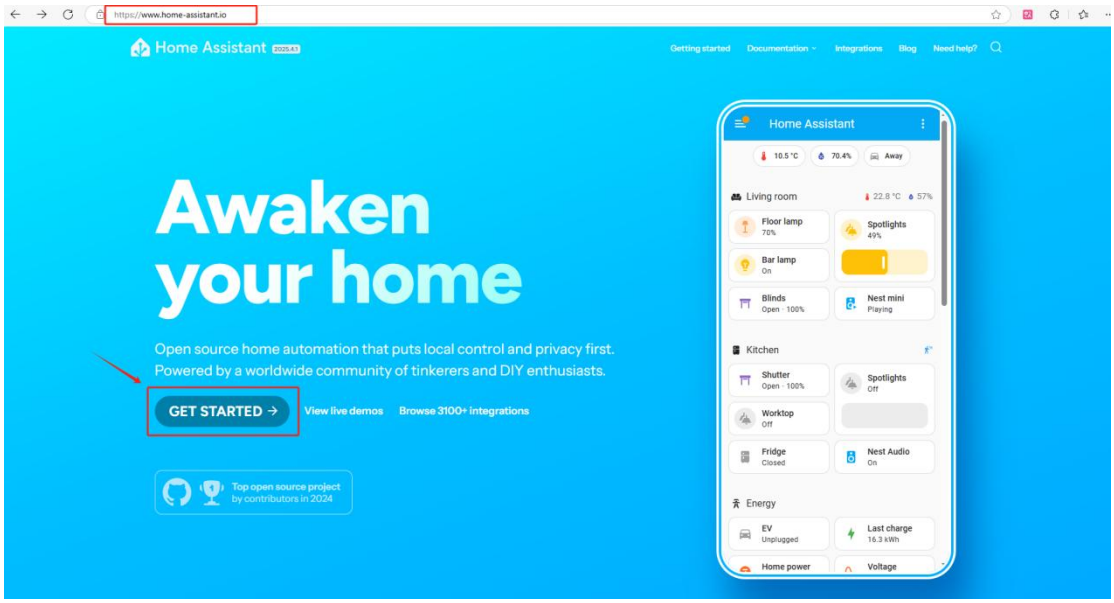
https://www.elecrow.com/esp32-display-7-inch-hmi-display-rgb-tft-lcd-touch-screen-support-lvgl.html
https://www.elecrow.com/esp32-display-5-inch-hmi-display-rgb-tft-lcd-touch-screen-support-lvgl.html

A humidity and temperature sensor;
purchase link :
https://www.elecrow.com/crowtail-dht20.html

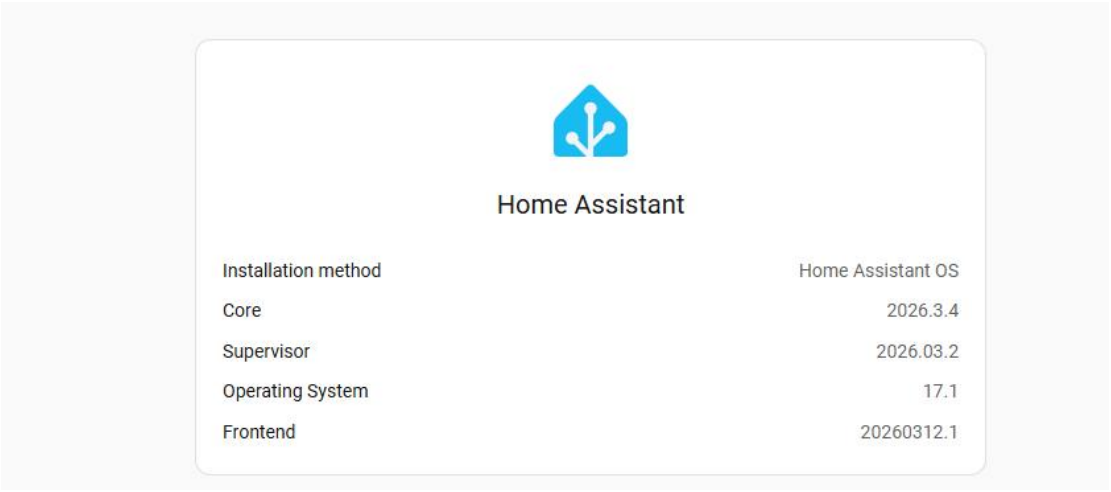
An LED light;
purchase link :
https://www.elecrow.com/crowtail-led-p-1224.html

Download HomeAssitant

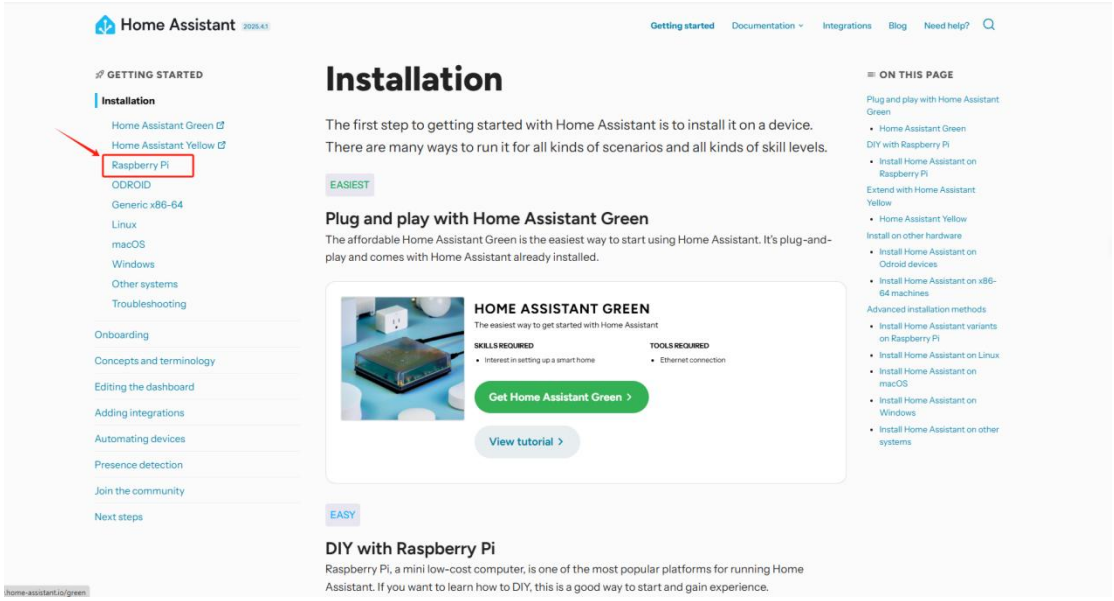
Open the official website of HomeAssitant：https://www.home-assistant.io/



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 (5.0-inch \ 7.0-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 (5.0-inch \ 7.0-inch);

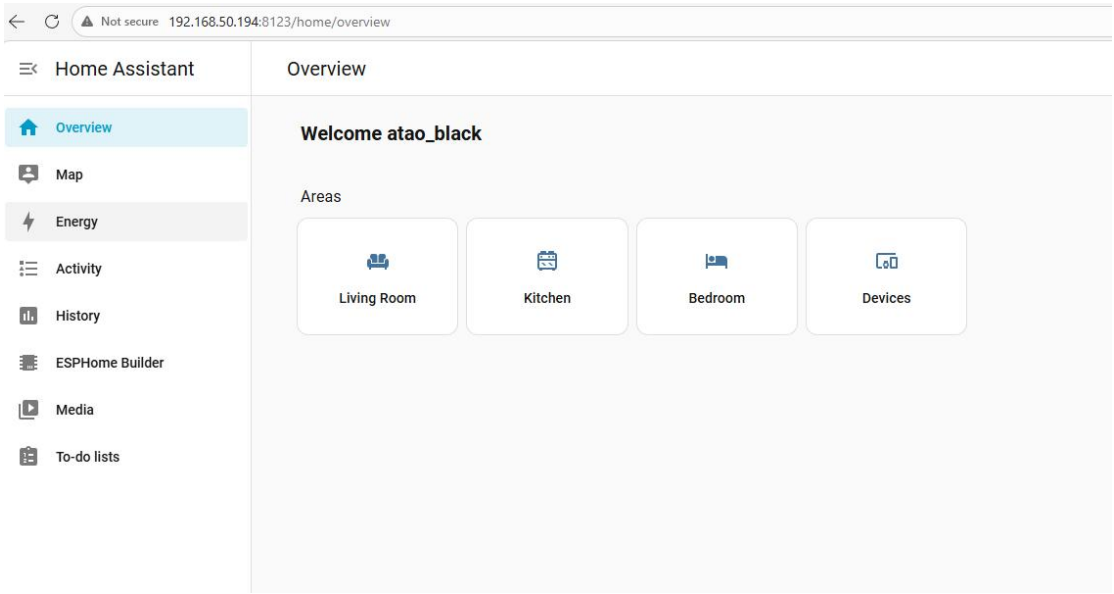
③ Raspberry Pi with the Home Assistant system

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.

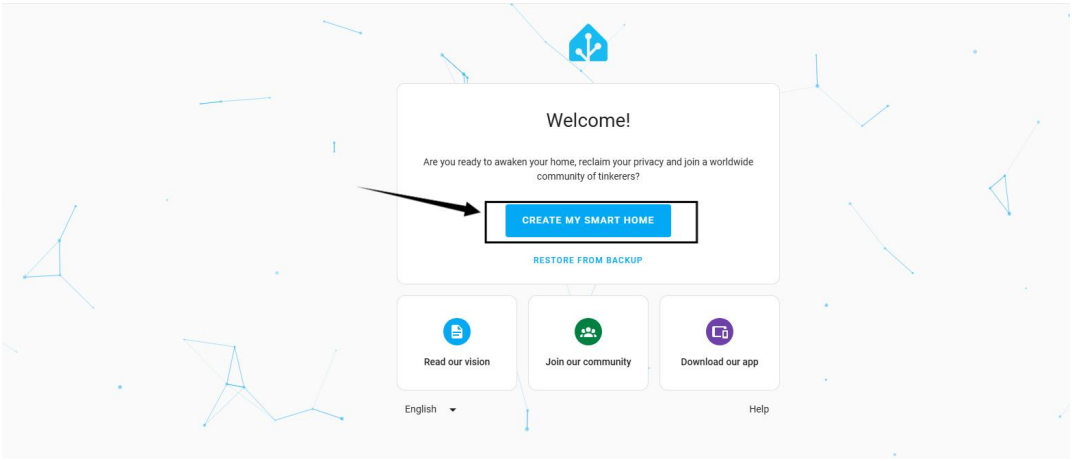
(You need to connect a display screen to the Raspberry Pi in order to see these information.)

Remember the IP address and the Home Assistant URL.

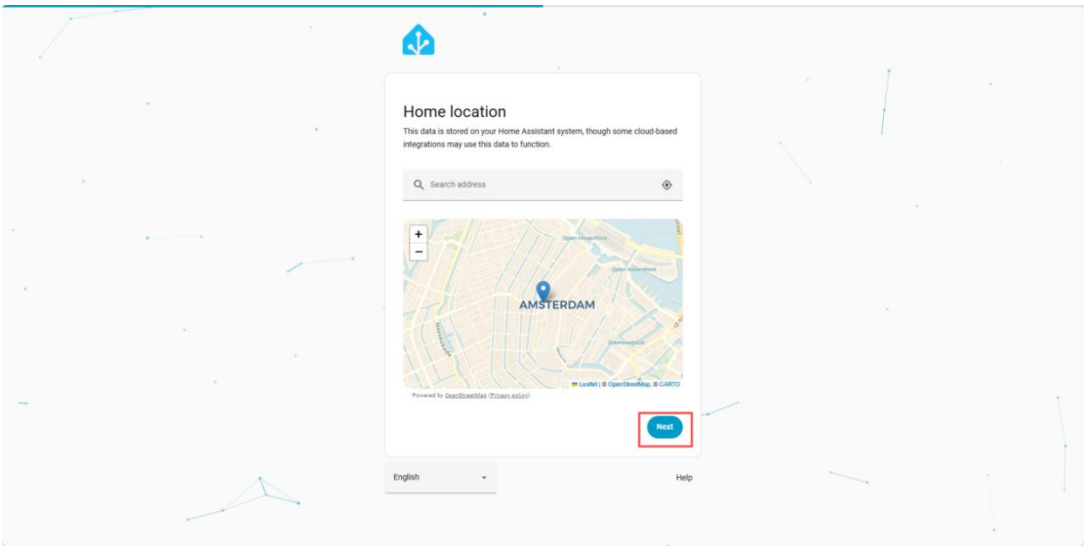
In my case, the IP address is: 192.168.50.194
So I enter **192.168.50.194:8123** in the browser.



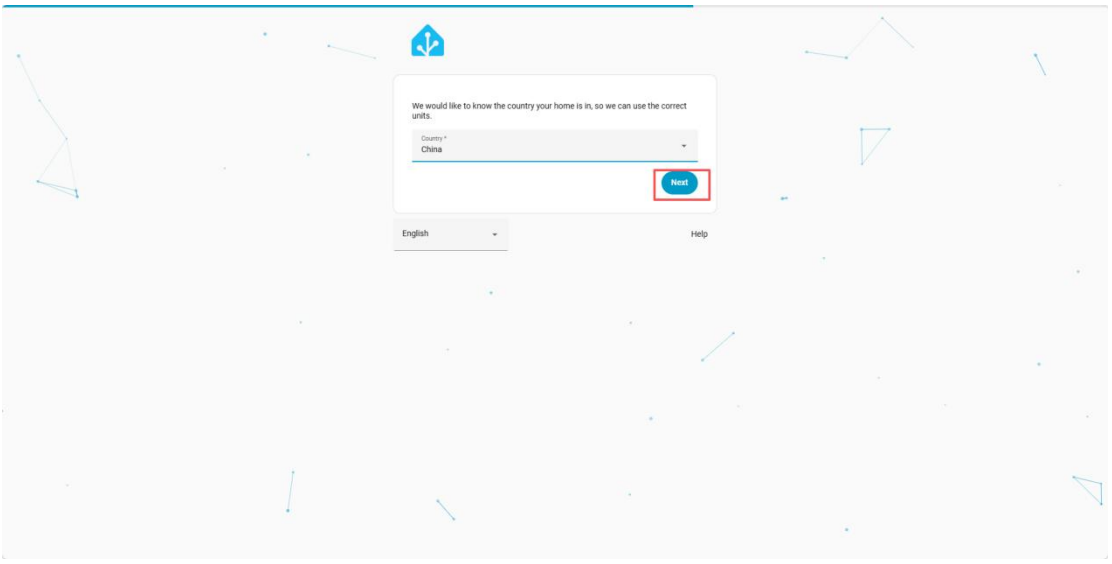
Next, start configuring the Home Assistant system.



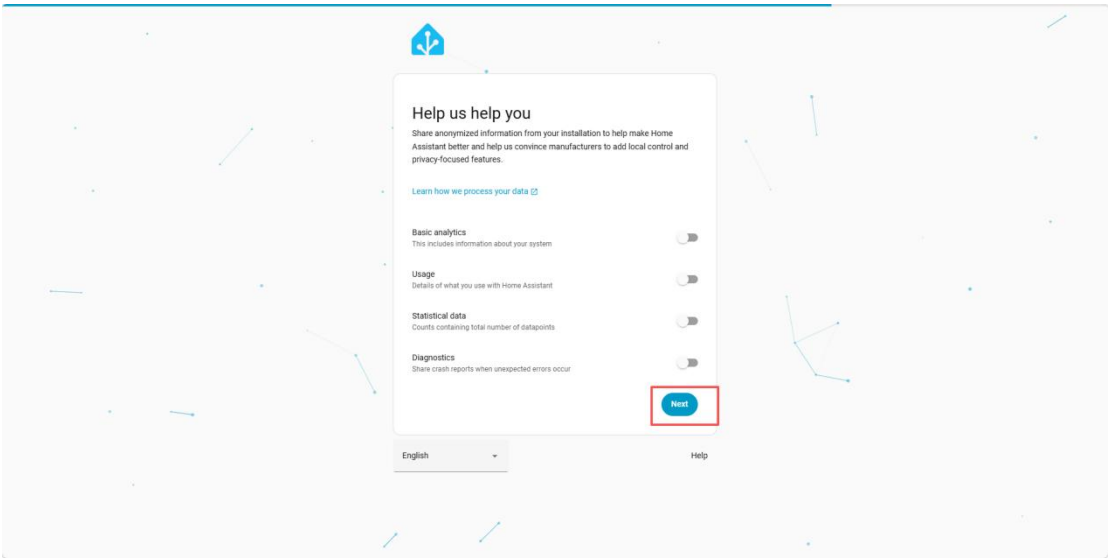
You can either manually set your location or allow it to be detected automatically.



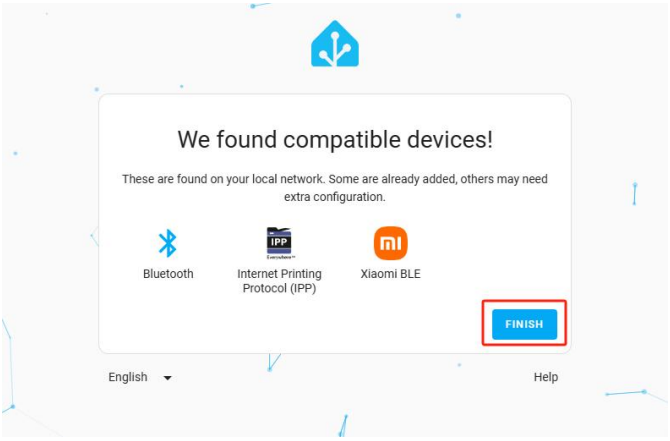
Set the country where you are located.



Keep it as default here.

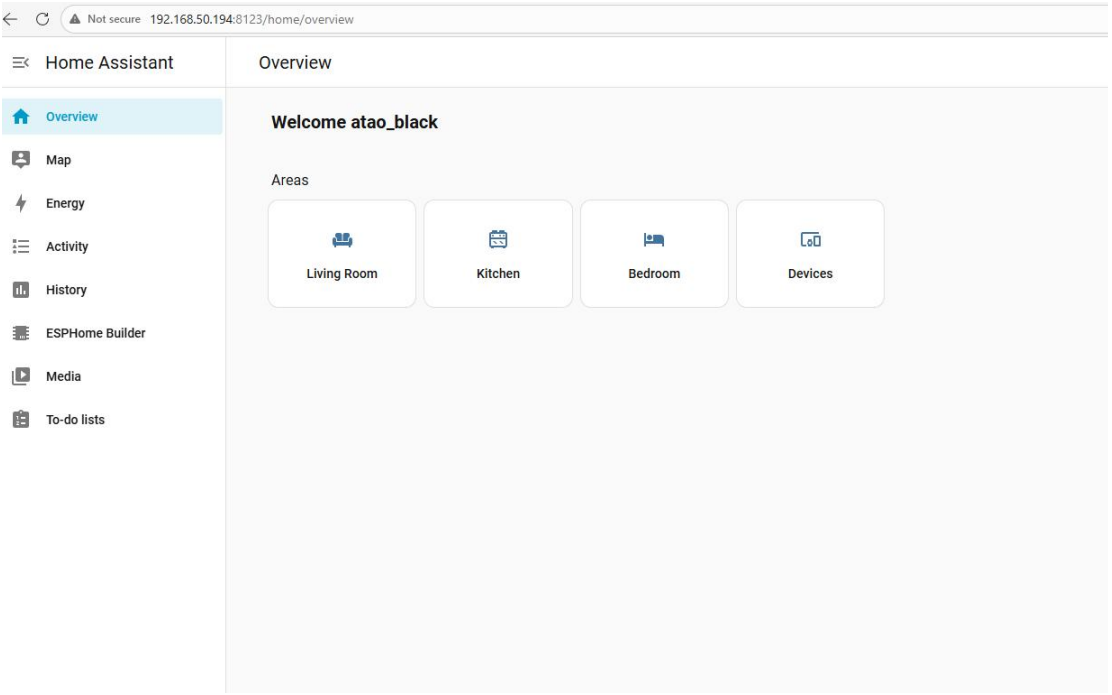


You can either add smart devices now or click **Finish** to add them later.
Here, we will add the devices later.

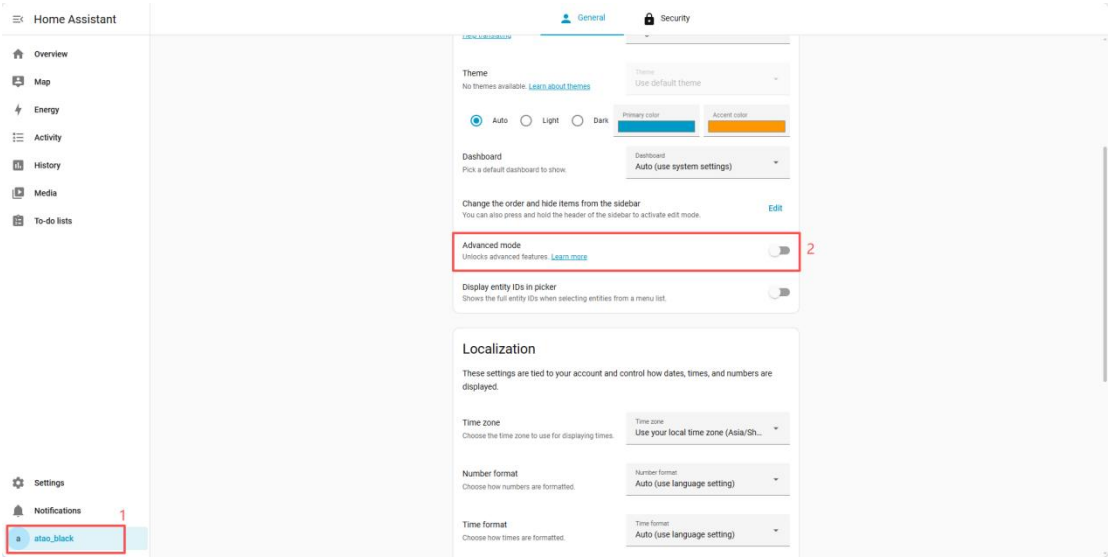


Add ESPHome

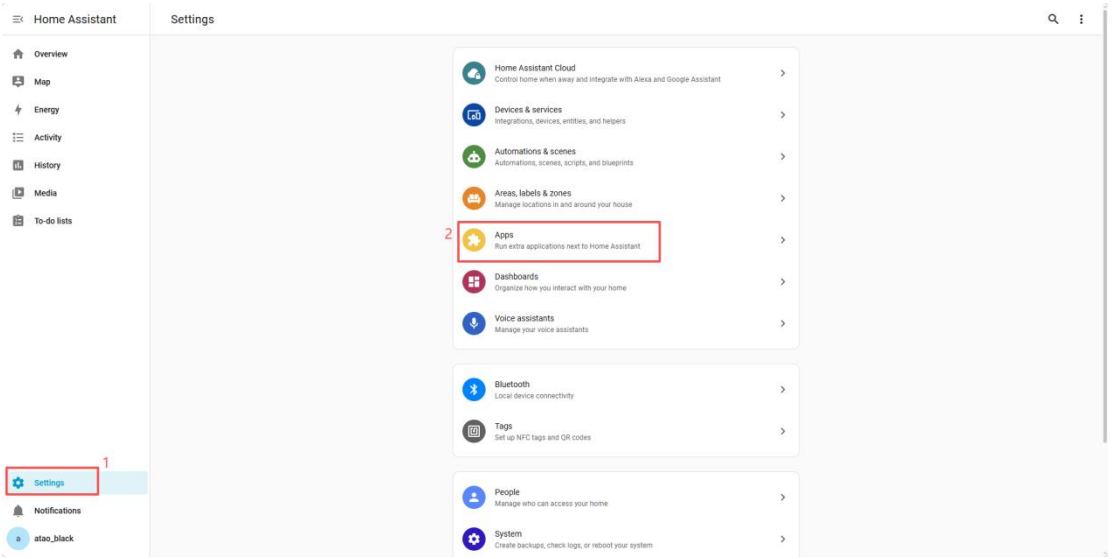
This will bring you to the main interface of Home Assistant.



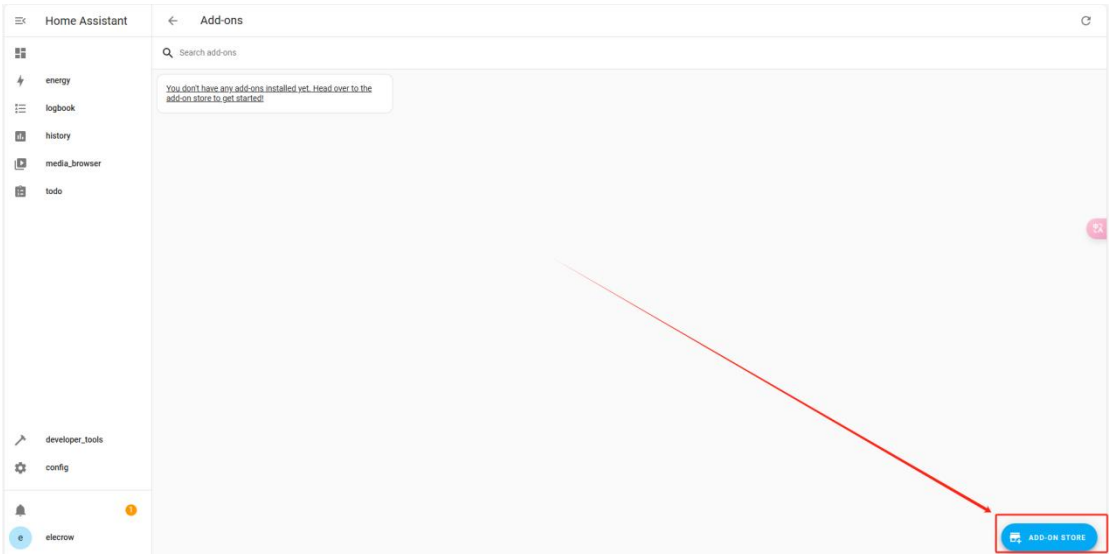
Click on the username and enable "Advanced Mode."

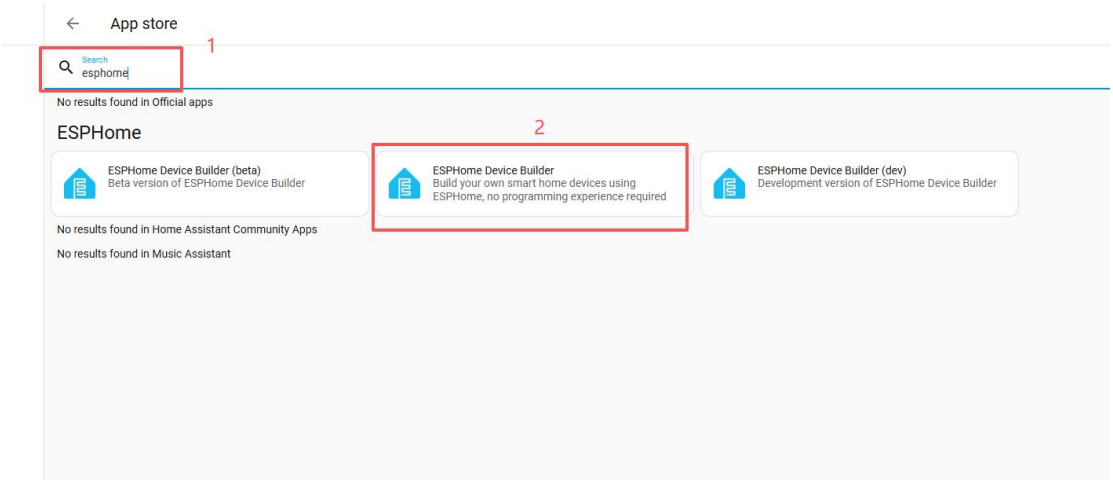


Click on "Settings" and then "Apps."

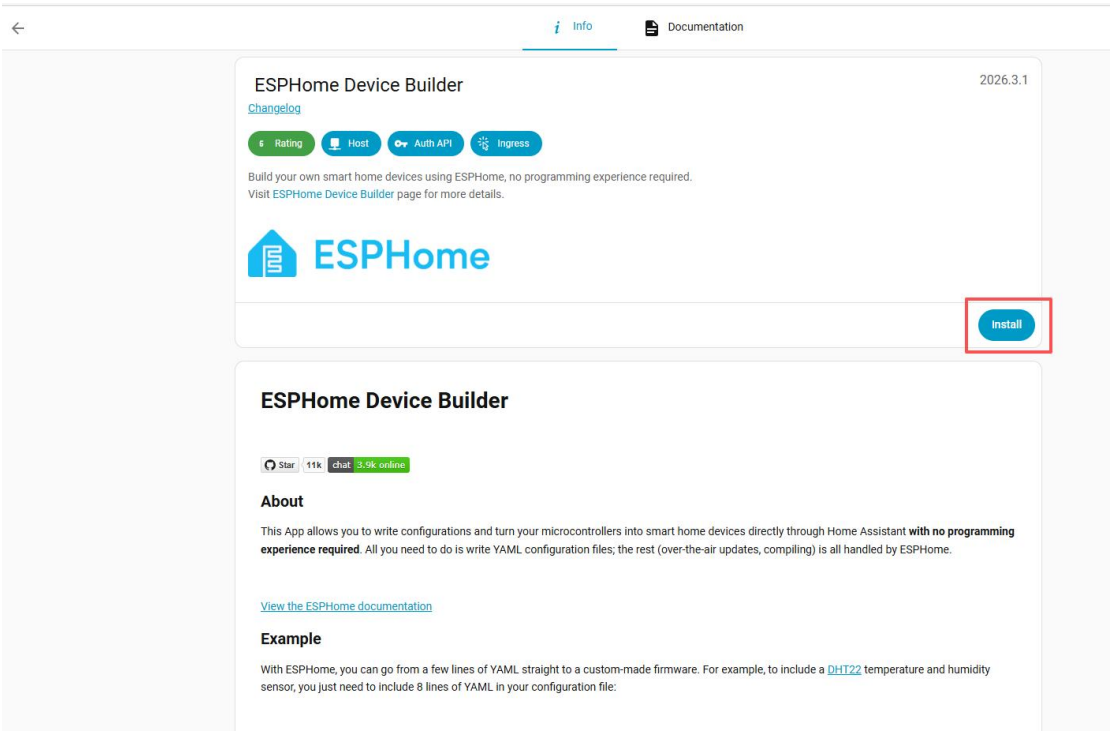


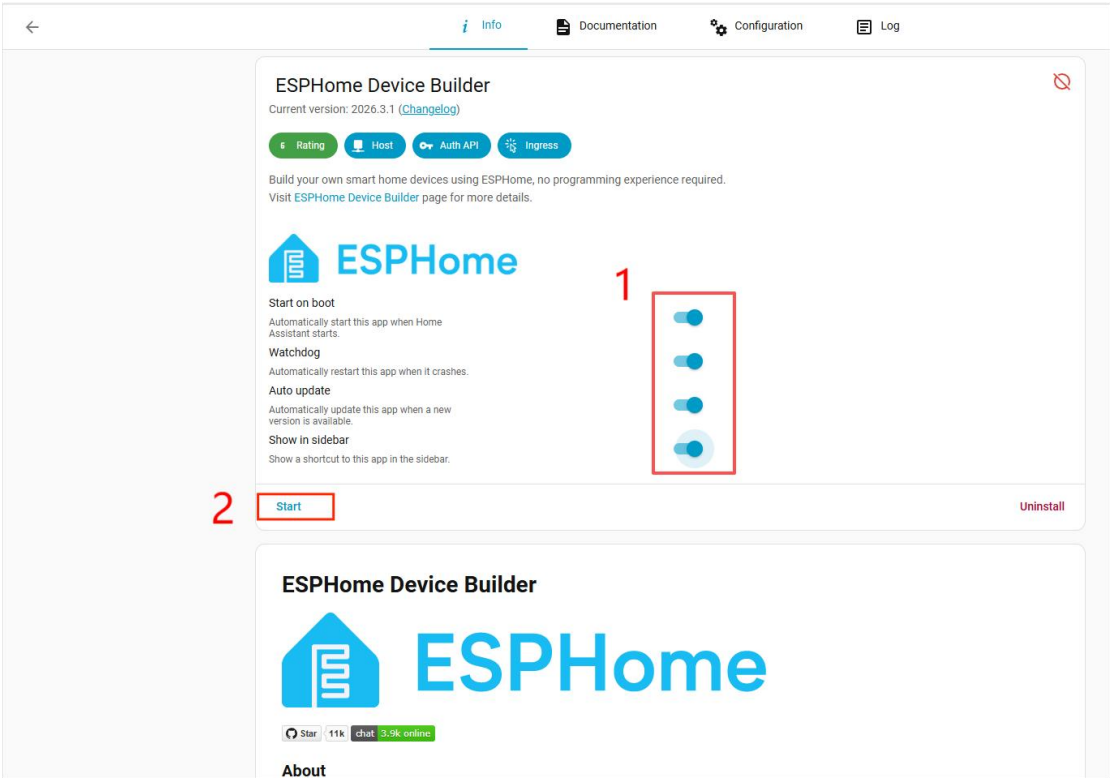
Click on "Apps" and type **ESPHome** to install it.





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



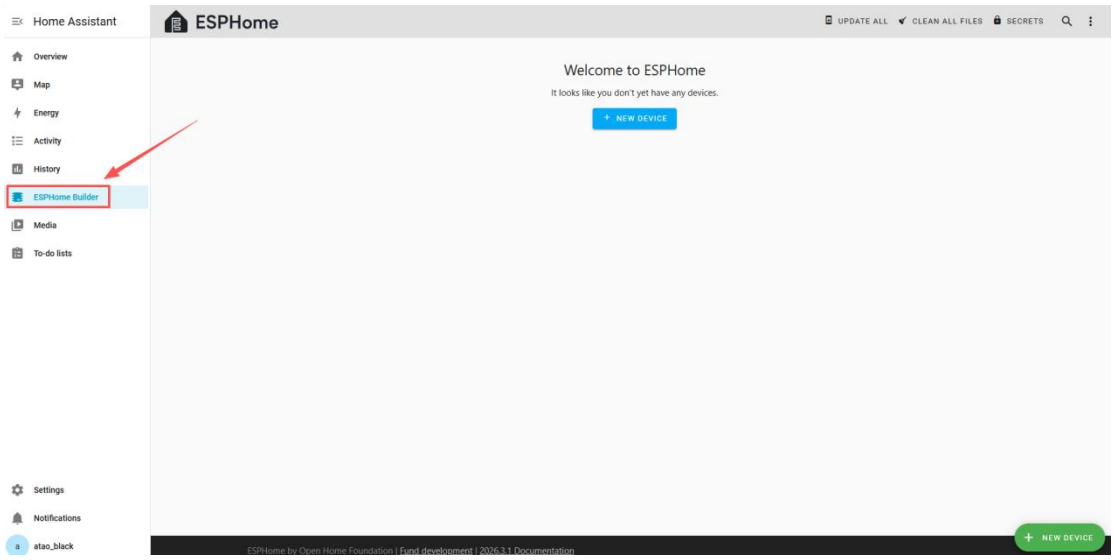


So we have successfully installed ESPHome. From now on, we can use ESPHome to edit the code and achieve the functions we want.

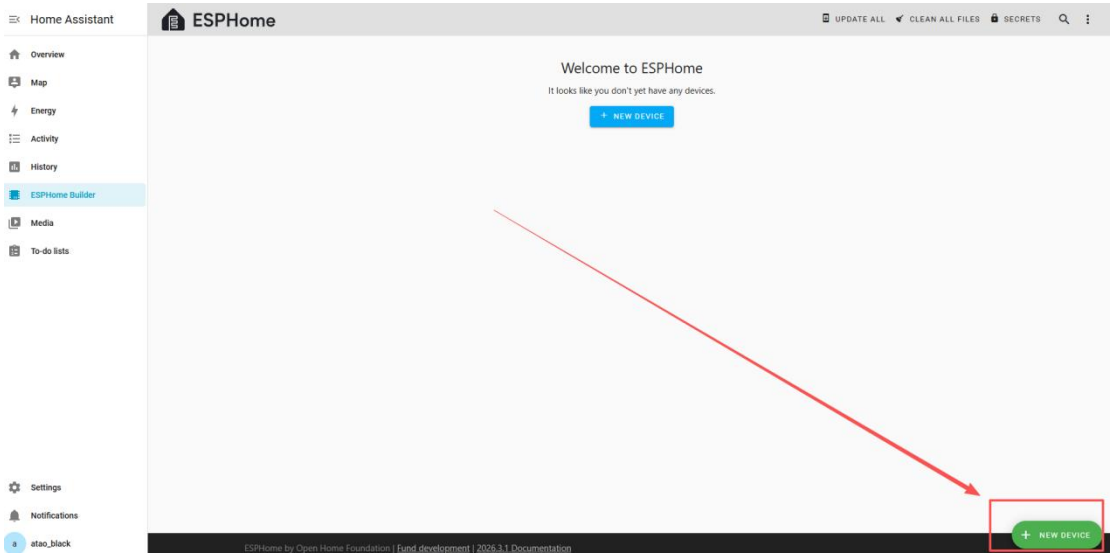
6. New Project

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

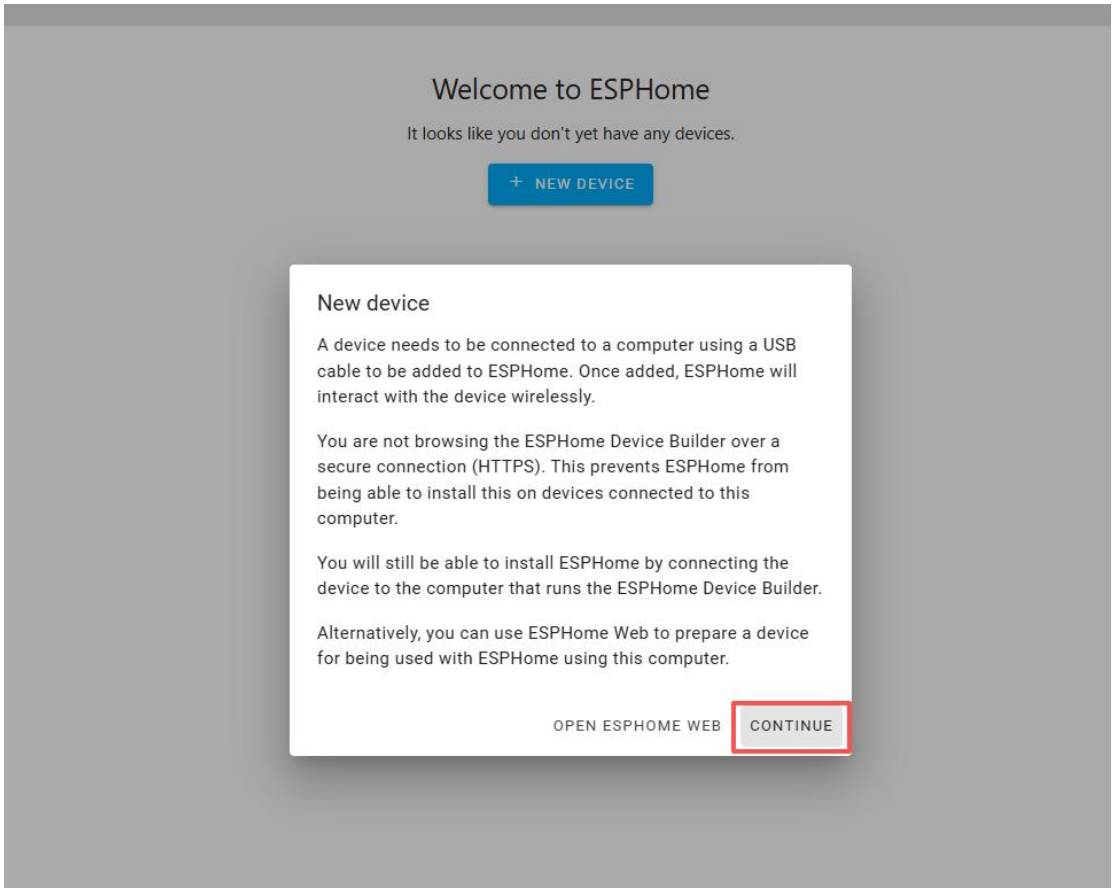
Come to ESPHome Builder



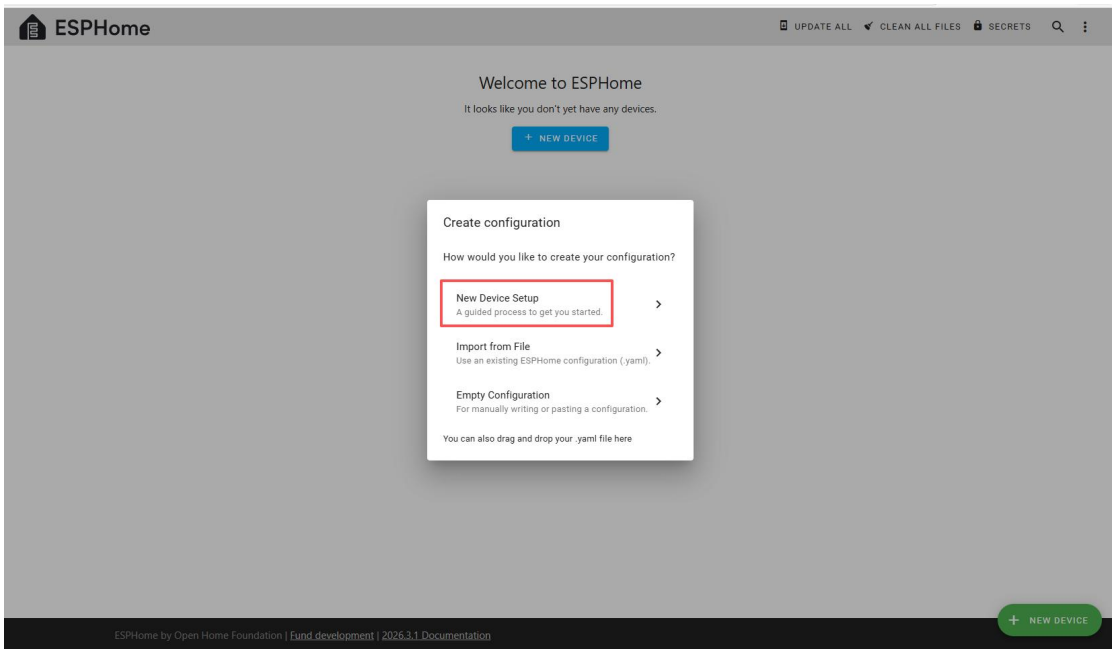
Click to add a new device



Click "Continue"



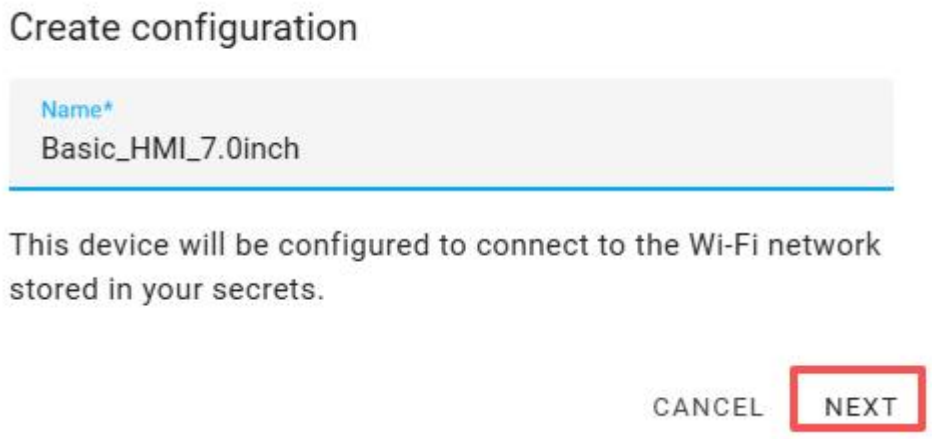
Add new equipment



Set a name for this project, as well as the name and password of the Wi-Fi that the equipment will connect to.

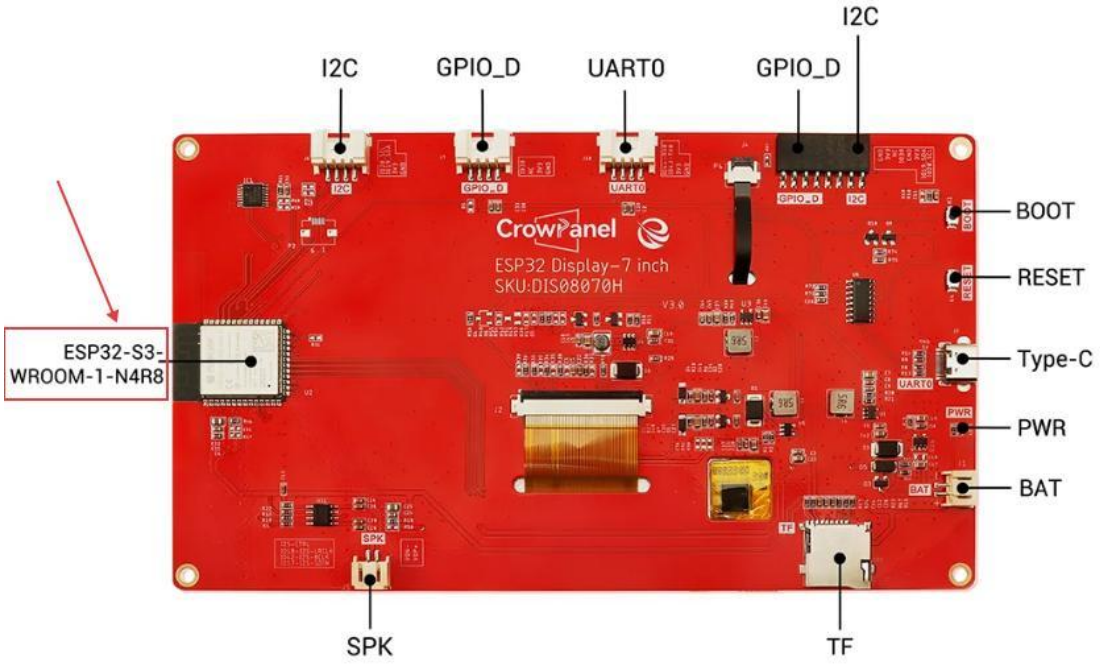
(The Wi-Fi must be within the same local area network as your computer host and ensure that your Wi-Fi operates on the 2.4GHz frequency band.)

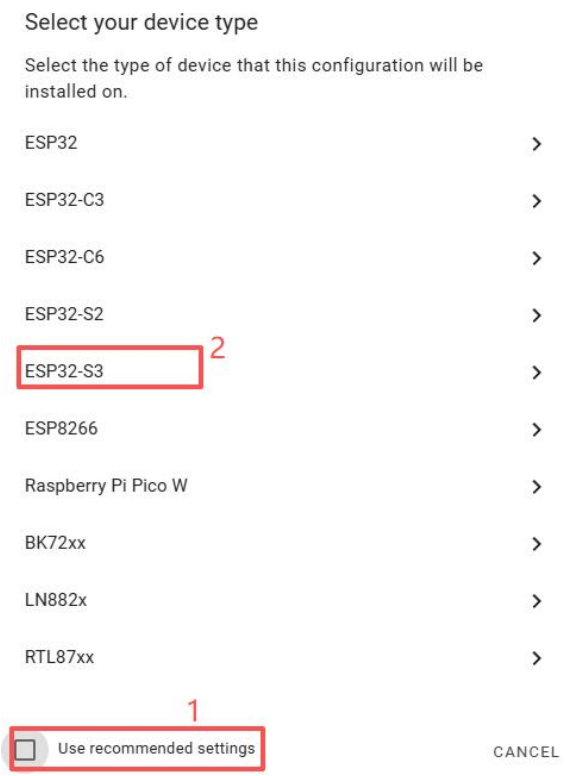
The naming here can be customized according to your own preferences, as long as it can clearly distinguish the current object being operated on. Next, I will take the 7.0-inch product as an example to demonstrate the entire configuration process and explain the related code. The operation steps for other size products are exactly the same. Finally, you just need to use the corresponding code provided by us to complete the driver.



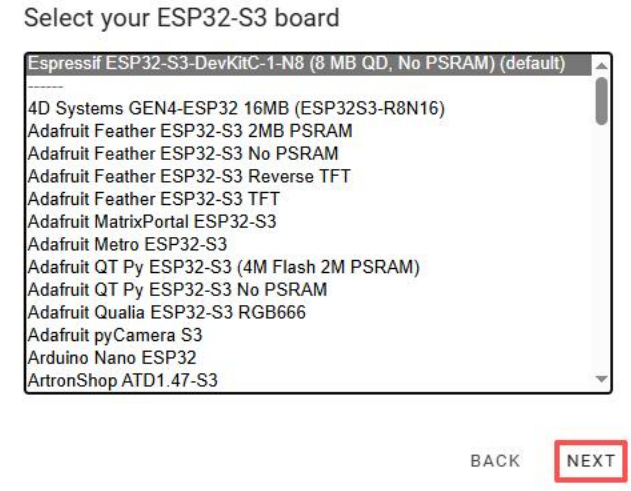
Here, do not check "Use recommended settings."

Next, based on the size of the CrowPanel Basic HMI ESP32 Display (5.0-inch / 7.0-inch) you are using, select the corresponding ESP32S3 main control chip.

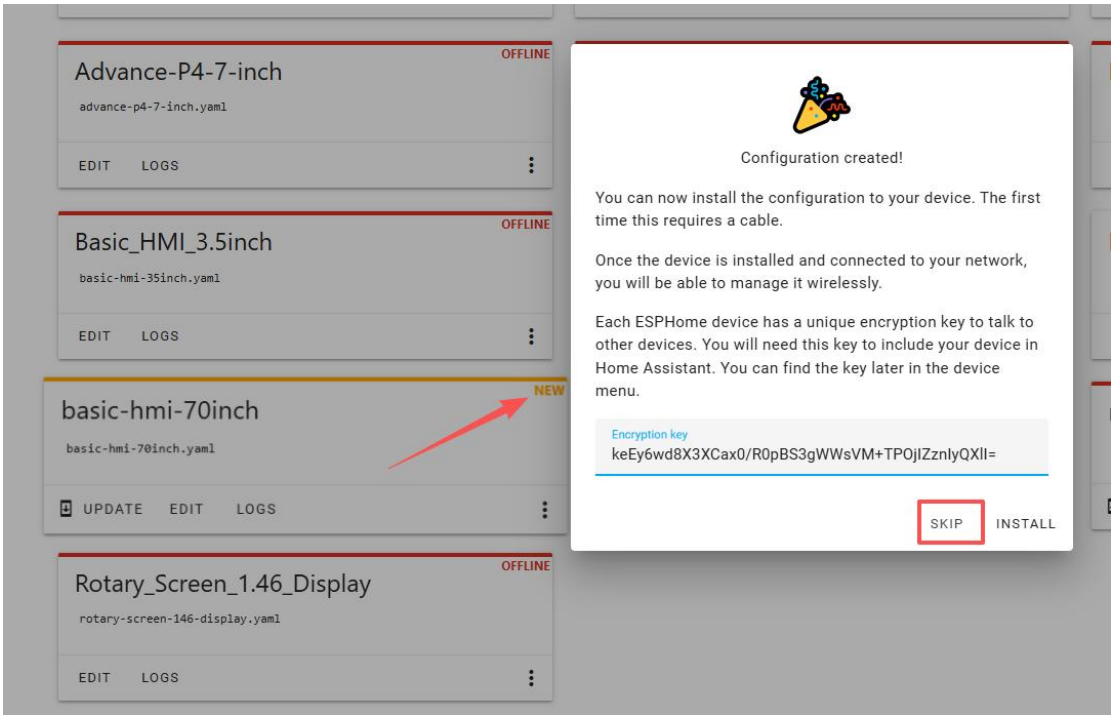




Next, choose any option (since we will replace it in the code later).

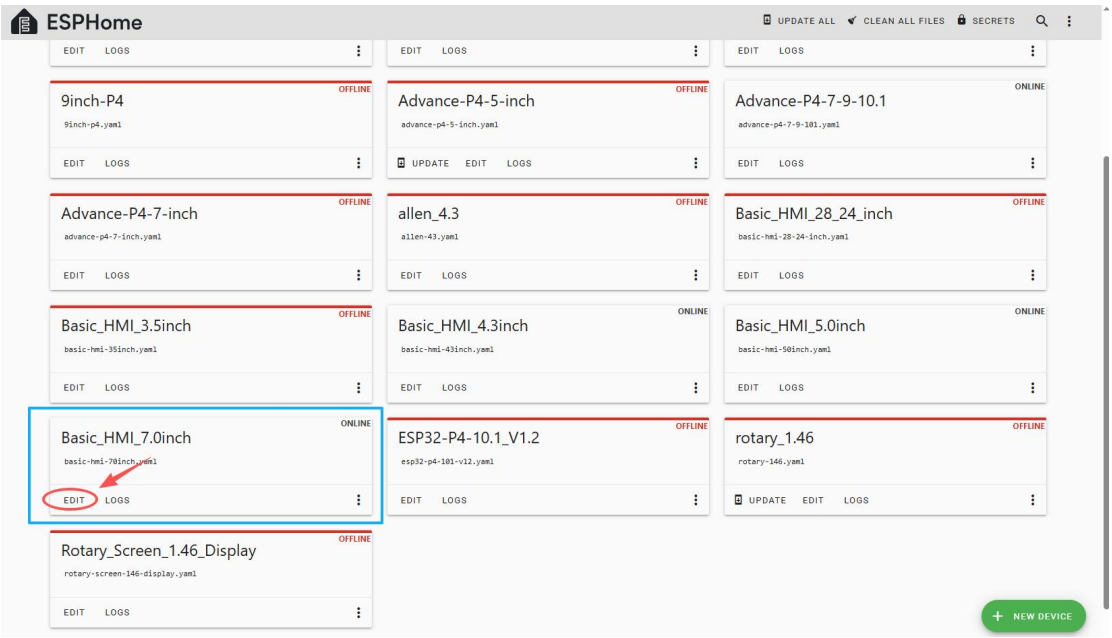


Here, click "SKIP".



7. Configuration Code

Then, return to the main interface, find the Basic_HMI_7.0inch 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.

basic-hmi-70inch.yaml

```yaml
esphome:
  name: basic-hmi-70inch
  friendly_name: Basic_HMI_7.0inch

esp32:
  board: esp32-s3-devkitc-1
  framework:
    type: esp-idf

# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "keEy6wd8X3XCax0/R0pBS3gWWsVM+TPOjIZznIyQXlI="

ota:
  - platform: esphome
    password: "eb82c0b3200dde55e78ffffda852e1db"

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Basic-Hmi-70Inch"
    password: "sGPi8gC5ATd2"

captive_portal:
```

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

7.0-Size Code:

https://github.com/Elecrow-RD/CrowPanel-7.0-HMI-ESP32-Display-800x480/tree/master/example/V3.0/ESPHome

5.0-Size Code:

https://github.com/Elecrow-RD/CrowPanel-5.0-HMI-ESP32-Display-800x480/tree/master/example/V3.0/ESPHome

Please Note: The core logic of this lesson is demonstrated using the 7.0-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-70inch.yaml

```yaml
esphome:
  name: basic-hmi-70inch
  friendly_name: Basic_HMI_7.0inch
  platformio_options:                          # Custom PlatformIO build configuration
    build_flags:
      - "-DBOARD_HAS_PSRAM"                    # Enable PSRAM support in compilation
    board_build.esp-idf.memory_type: qio_opi   # Set memory type for ESP-IDF framework
    board_build.flash_mode: dio                # Set SPI flash access mode to DIO

esp32:                                         # ESP32 specific hardware configuration
  board: esp32s3box                            # Base board model for ESP32-S3
  framework:
    type: esp-idf                              # Use official ESP-IDF framework, not Arduino
    sdkconfig_options:
      CONFIG_ESP32S3_DEFAULT_CPU_FREQ_240: y   # Set CPU frequency to max (240MHz)
      CONFIG_ESP32S3_DATA_CACHE_64KB: y        # Allocate 64KB for data cache
      CONFIG_SPIRAM_FETCH_INSTRUCTIONS: y      # Allow instruction fetch from SPIRAM
      CONFIG_SPIRAM_RODATA: y                  # Store read-only data in SPIRAM
      CONFIG_SPIRAM_USE: y                     # Enable external SPIRAM globally
      CONFIG_SPIRAM_SPEED_80M: y               # Set SPIRAM clock speed to 80MHz
      CONFIG_SPIRAM_MODE_OCT: y                # Use Octal mode for faster RAM access
      CONFIG_ESP32S3_SPIRAM_SUPPORT: y         # Enable ESP32-S3 specific SPIRAM support
  variant: esp32s3                             # Specify the chip variant exactly

psram:                                         # Physical PSRAM configuration block
  mode: octal                                  # Octal SPI mode matches board layout
  speed: 80MHz                                 # 80MHz speed matches sdkconfig options


# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "keEy6wd8X3XCax0/R0pBS3gWWsVM+TPOjIZznIyQXlI="

ota:
  - platform: esphome
    password: "eb82c0b3200dde55e78ffffda852e1db"
```

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.

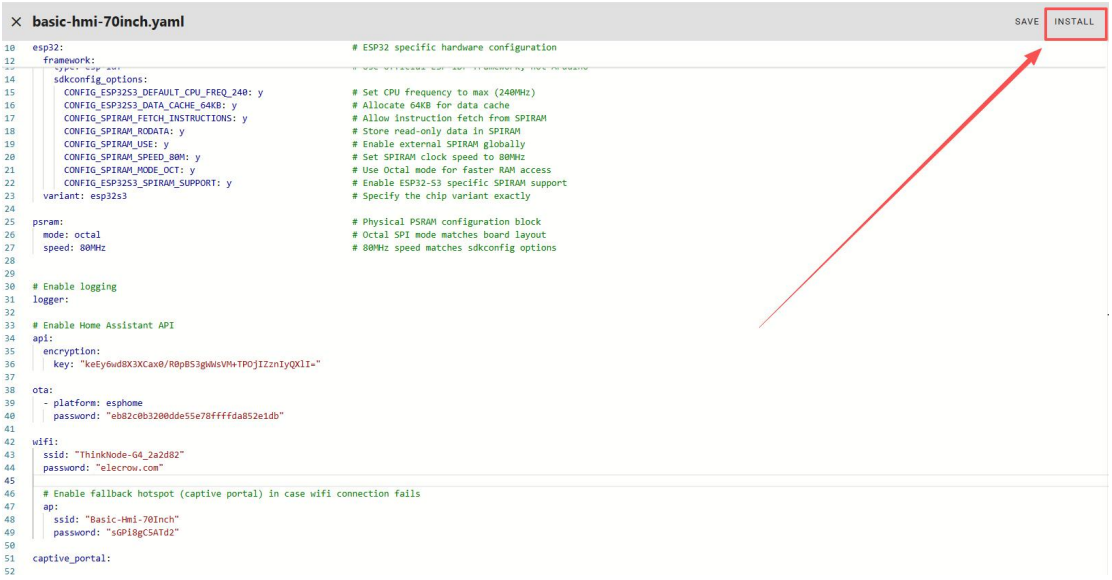
Keep them all within the same local area network.

basic-hmi-70inch.yaml

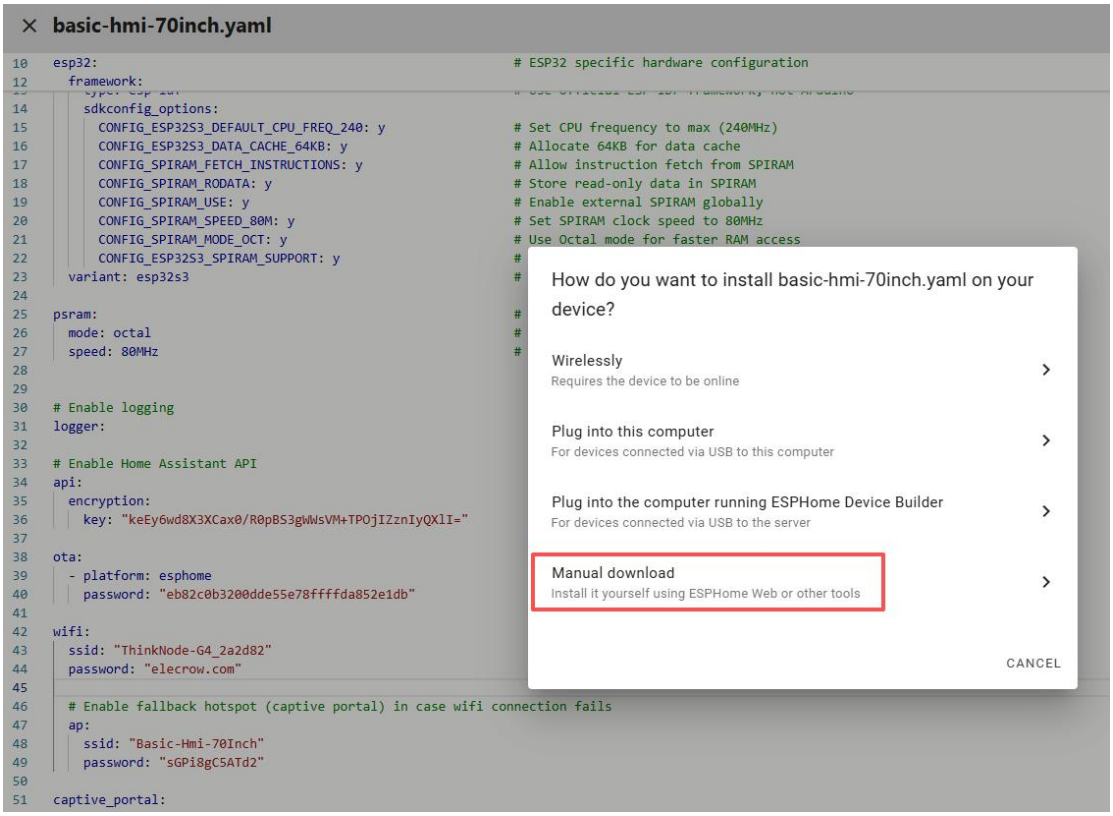
```yaml
esp32:                                         # ESP32 specific hardware configuration
  framework:
    sdkconfig_options:
      CONFIG_ESP32S3_DEFAULT_CPU_FREQ_240: y   # Set CPU frequency to max (240MHz)
      CONFIG_ESP32S3_DATA_CACHE_64KB: y        # Allocate 64KB for data cache
      CONFIG_SPIRAM_FETCH_INSTRUCTIONS: y      # Allow instruction fetch from SPIRAM
      CONFIG_SPIRAM_RODATA: y                  # Store read-only data in SPIRAM
      CONFIG_SPIRAM_USE: y                     # Enable external SPIRAM globally
      CONFIG_SPIRAM_SPEED_80M: y               # Set SPIRAM clock speed to 80MHz
      CONFIG_SPIRAM_MODE_OCT: y                # Use Octal mode for faster RAM access
      CONFIG_ESP32S3_SPIRAM_SUPPORT: y         # Enable ESP32-S3 specific SPIRAM support
  variant: esp32s3                             # Specify the chip variant exactly

psram:                                         # Physical PSRAM configuration block
  mode: octal                                  # Octal SPI mode matches board layout
  speed: 80MHz                                 # 80MHz speed matches sdkconfig options


# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "keEy6wd8X3XCax0/R0pBS3gWWsVM+TPOjIZznIyQXlI="

ota:
  - platform: esphome
    password: "eb82c0b3200dde55e78ffffda852e1db"

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

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Basic-Hmi-70Inch"
    password: "sGPi8gC5ATd2"

captive_portal:
```

8. First upload of code

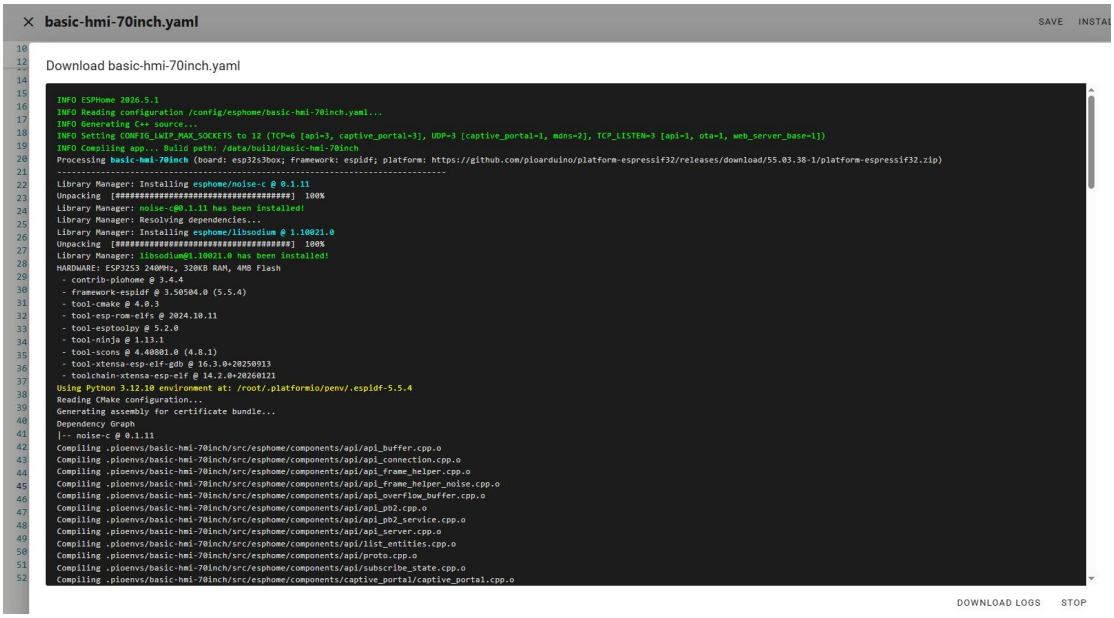
Once the code replacement is complete, click "INSTALL" in the top right corner.



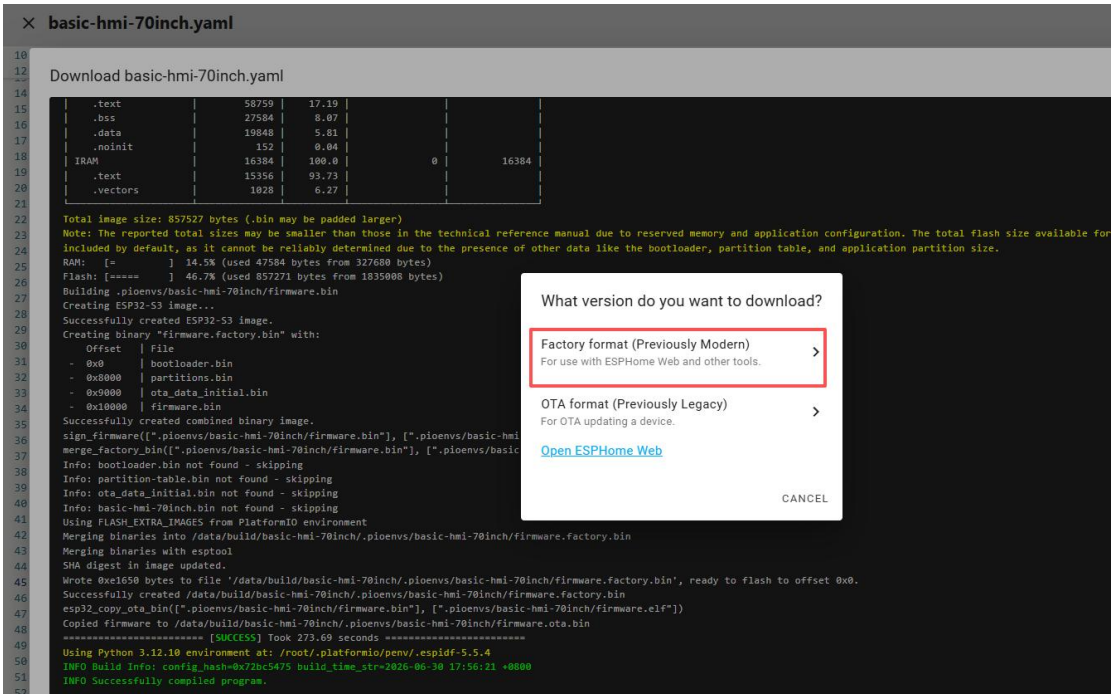
Select "Manual download".



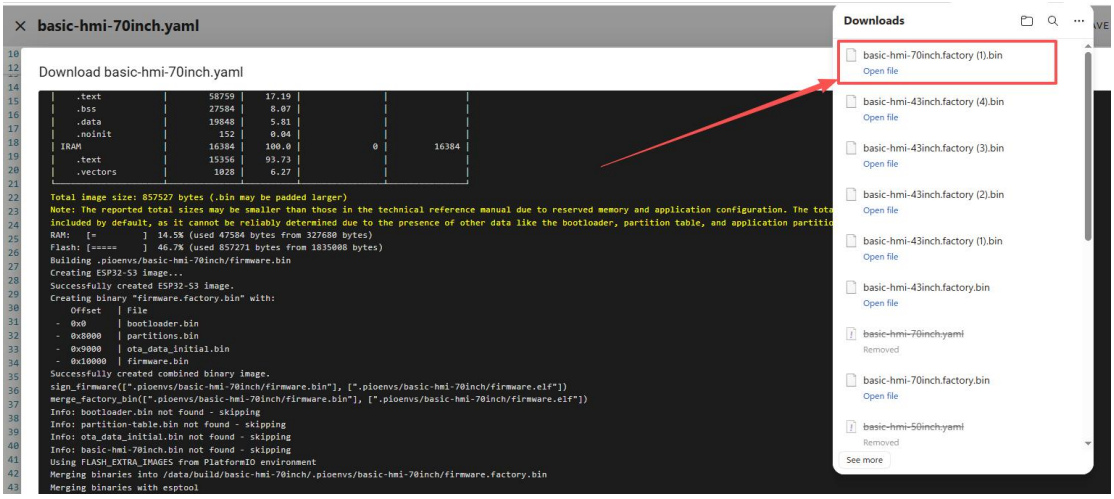
Wait for a few minutes until the installation is complete.The first installation and download process may take some time as it involves downloading necessary tools and libraries and compiling the code.



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



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



Remember the path of this .bin file.

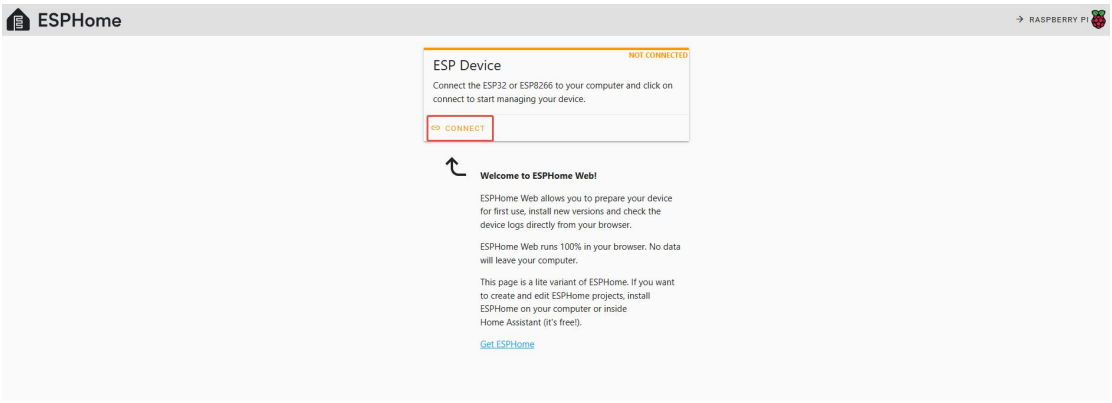
Open the following website:

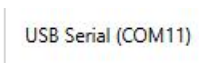
https://web.esphome.io/?dashboard_wizard

Next, we will flash this .bin file into the CrowPanel Basic HMI ESP32 Display (5.0-inch / 7.0-inch) .

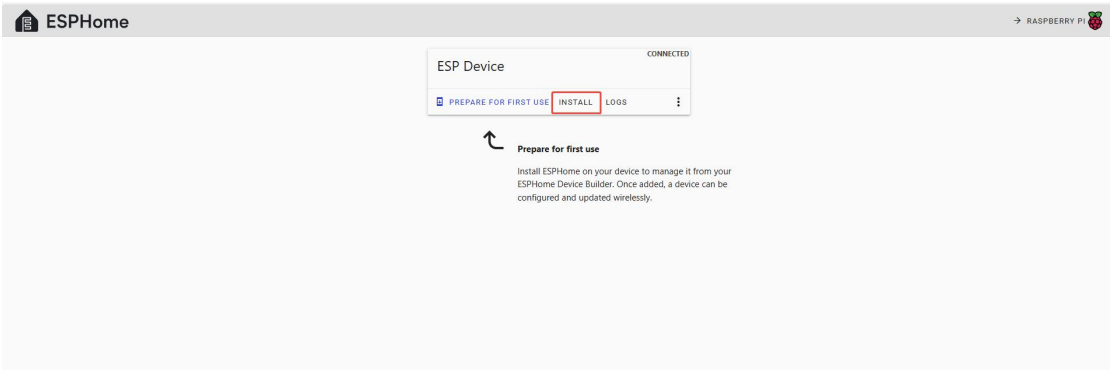
Connect the CrowPanel Basic HMI ESP32 Display (5.0-inch / 7.0-inch) to your computer.

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

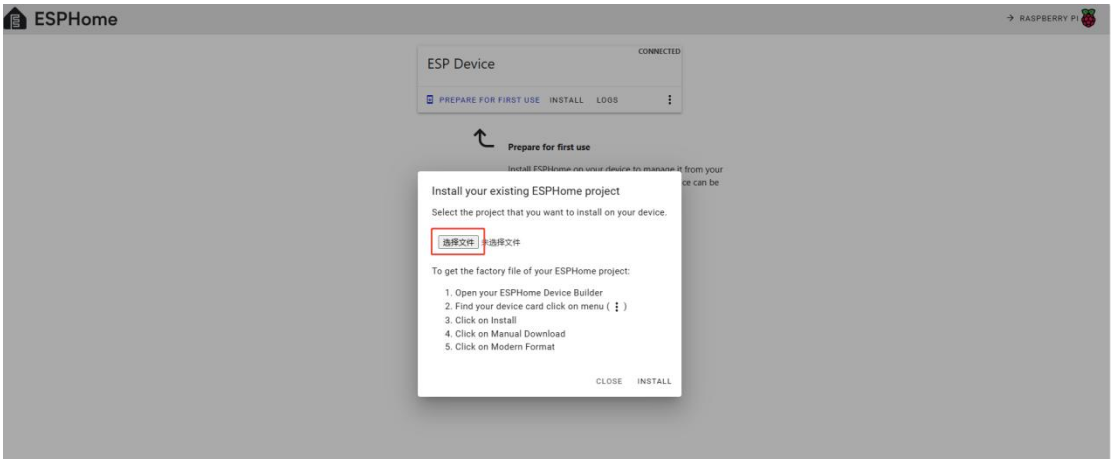


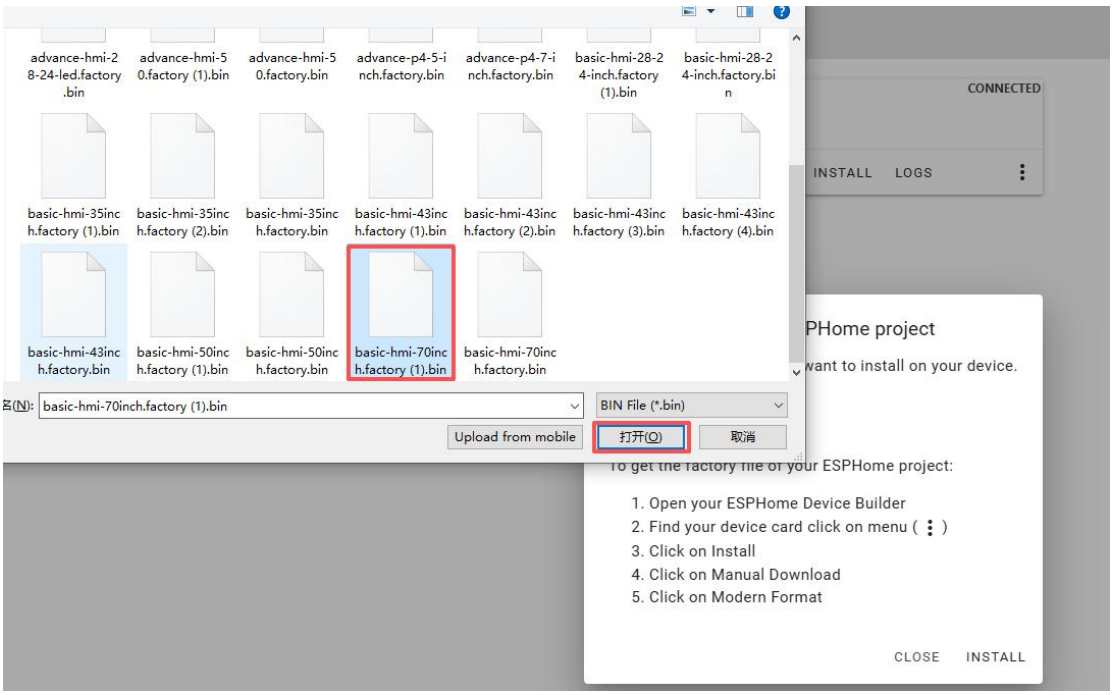


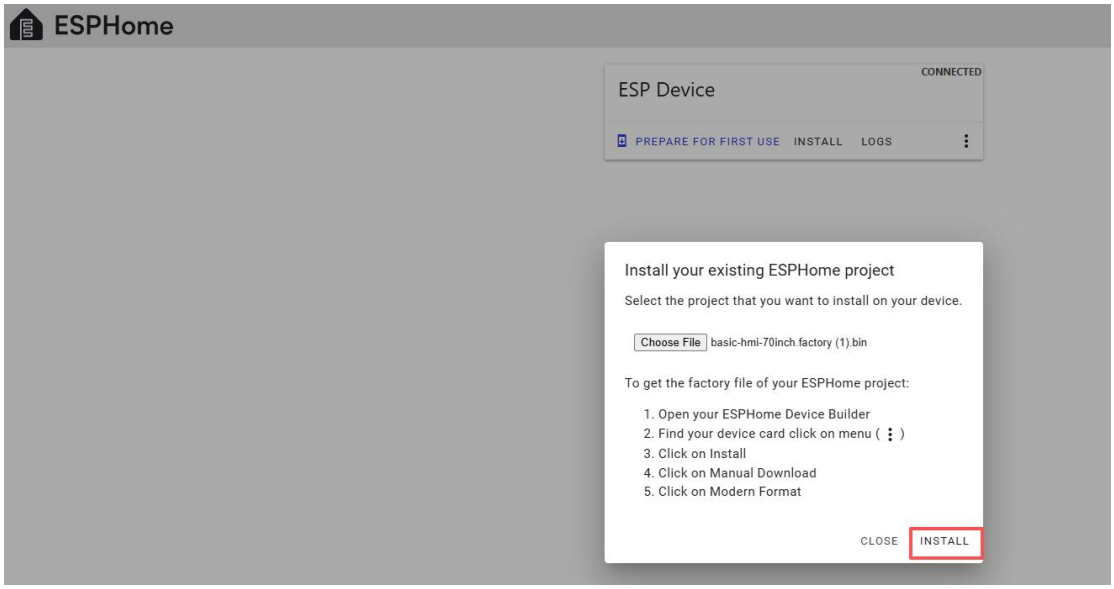
After connecting the CrowPanel Basic HMI ESP32 Display (5.0-inch / 7.0-inch) ,click "Install".



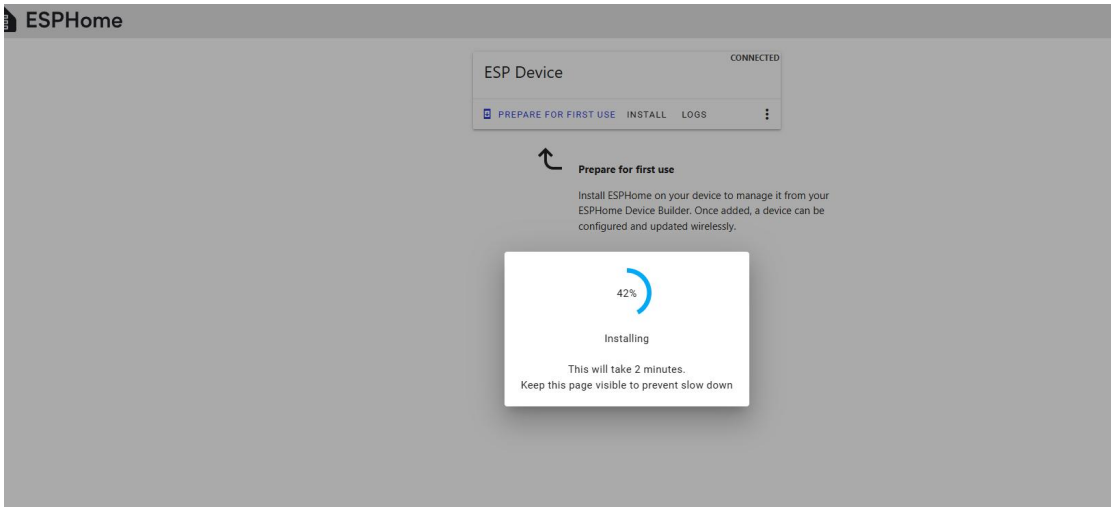
Add the .bin file you just downloaded, then click "Install".



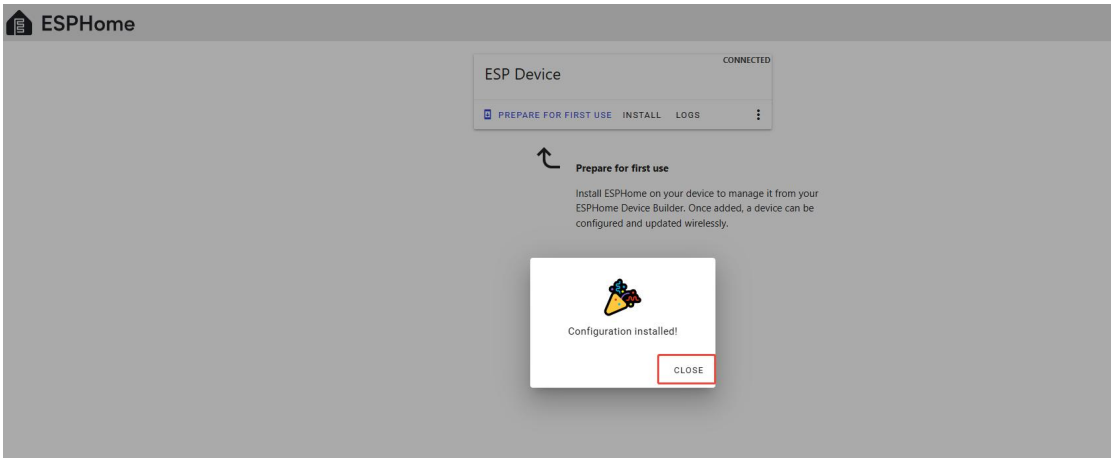




Wait for a few minutes.

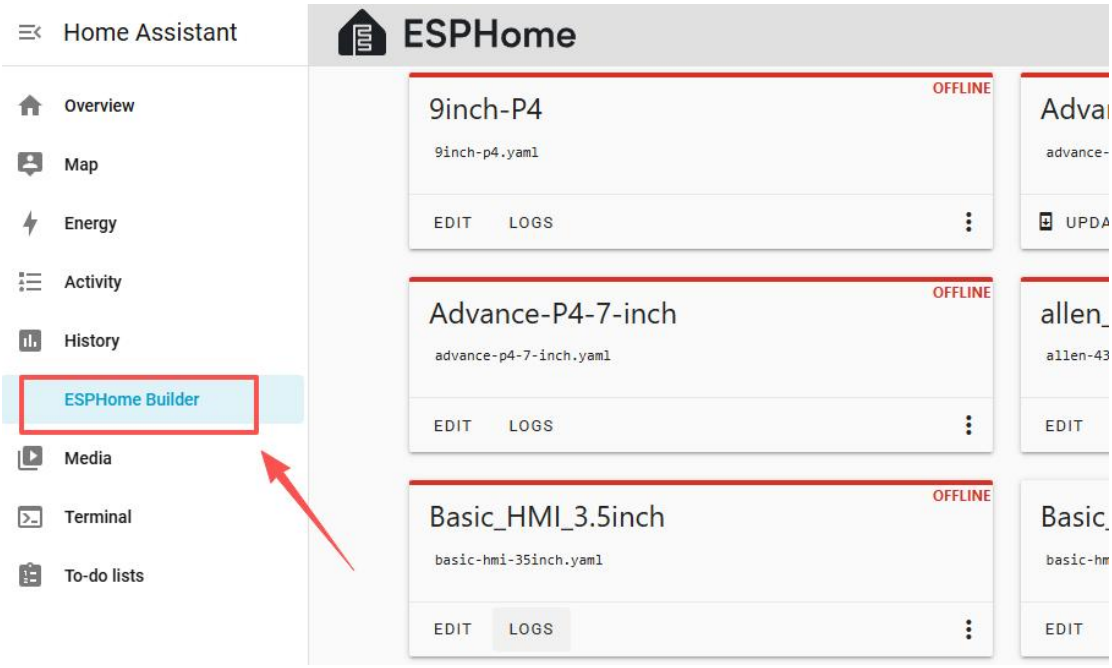


After the installation is complete, click "Close".

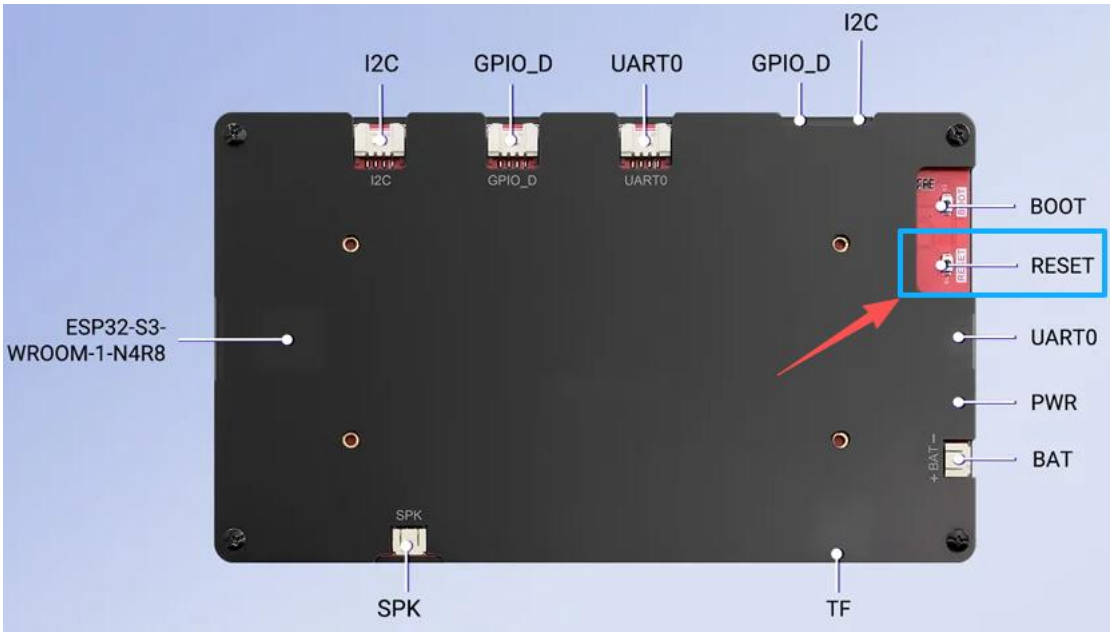


9. Observe the Wi-Fi connection status of the equipment

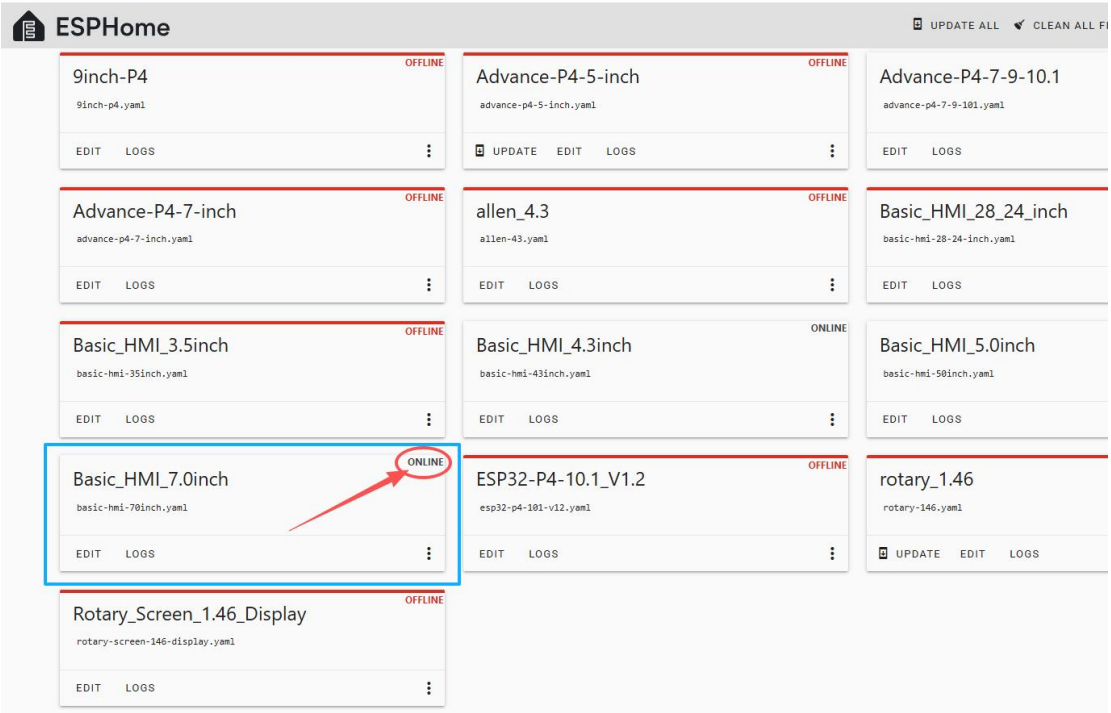
After successfully flashing the .bin file, return to the ESPHome page in Home Assistant.



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 (5.0-inch / 7.0-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.

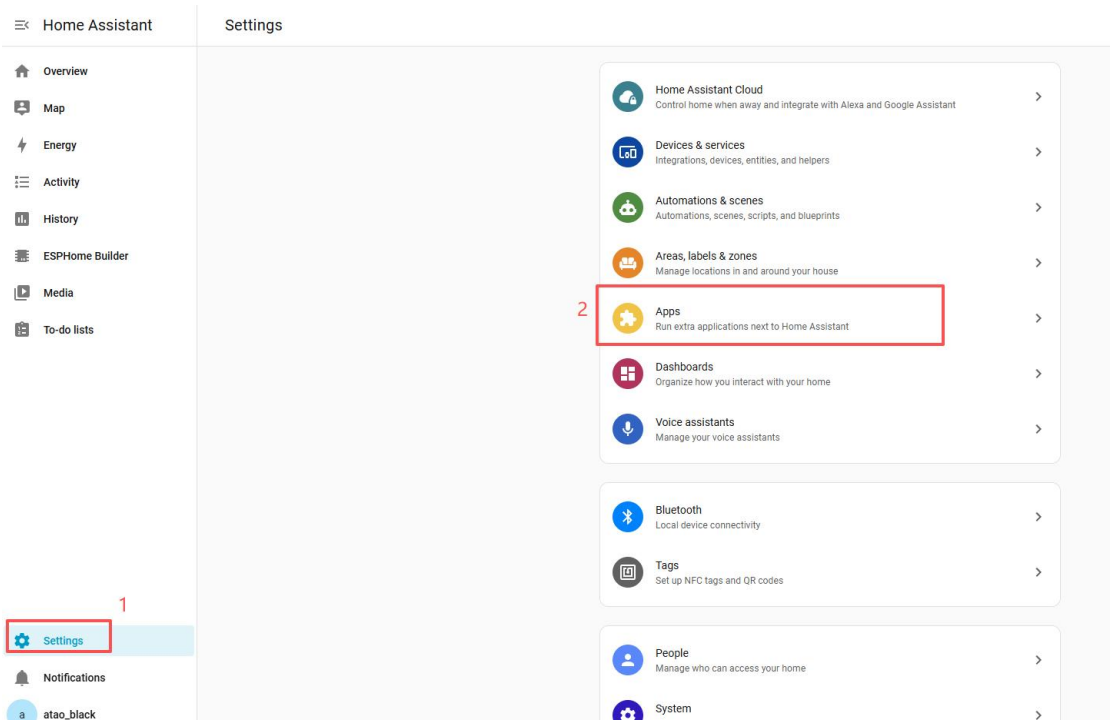
10.upload pictures

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.

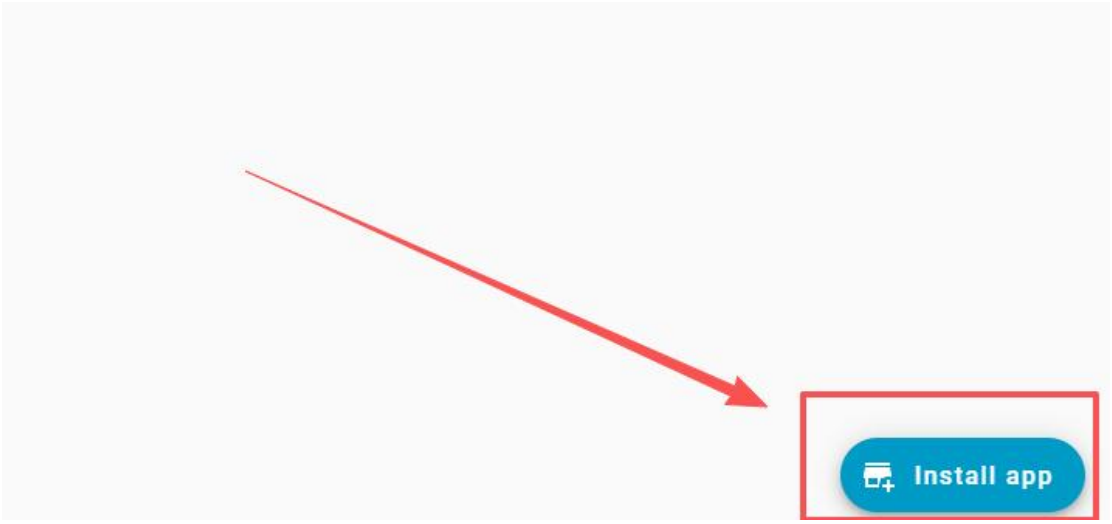
Click here to download the materials shown on our screen.
https://github.com/Elecrow-RD/CrowPanel-7.0-HMI-ESP32-Display-800x480/tree/master/example/V3.0/ESPHome/Materials

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

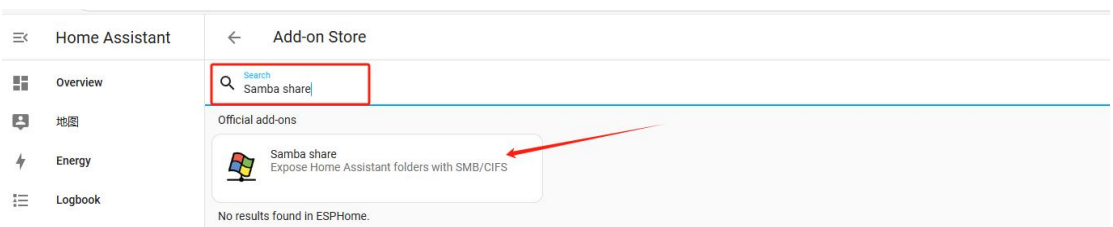
Click Settings and select Apps



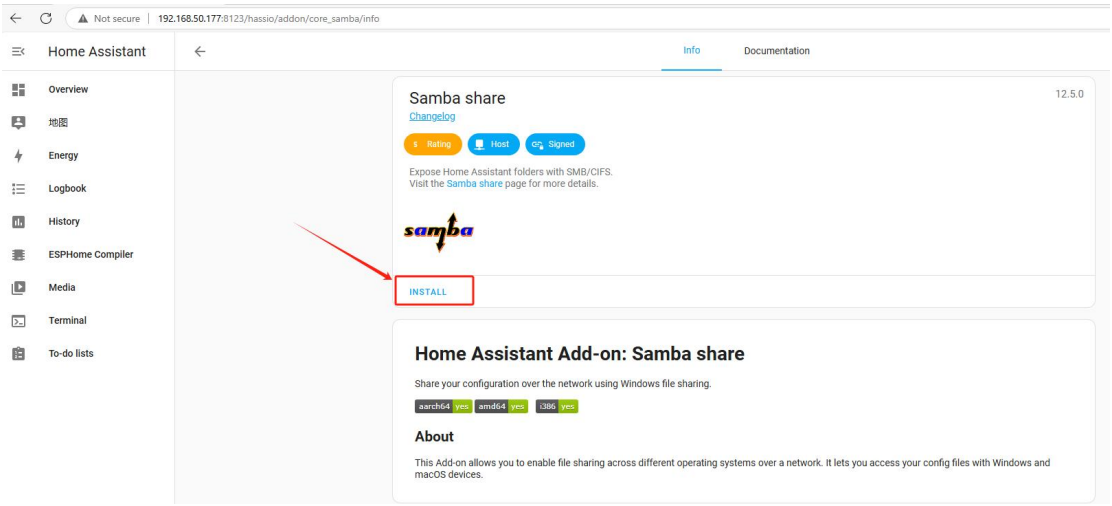
Then, click the “Install app” at the bottom right corner.



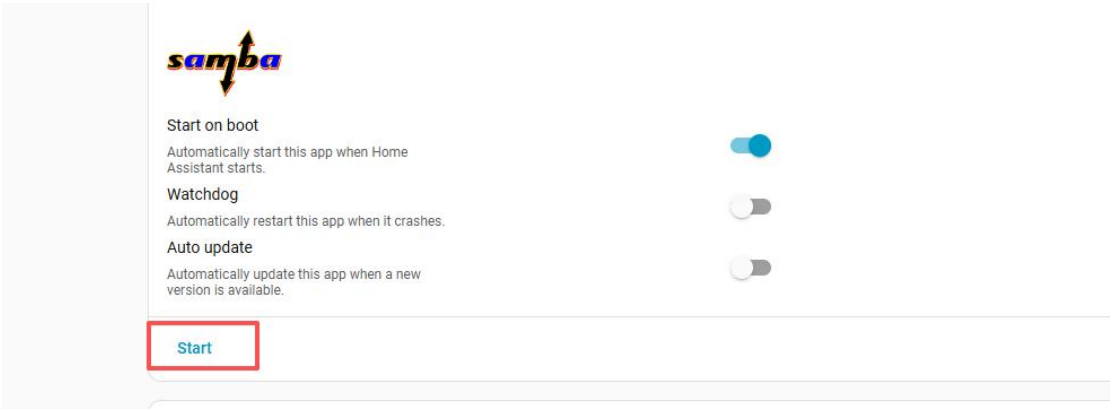
Search for Samba share at the top



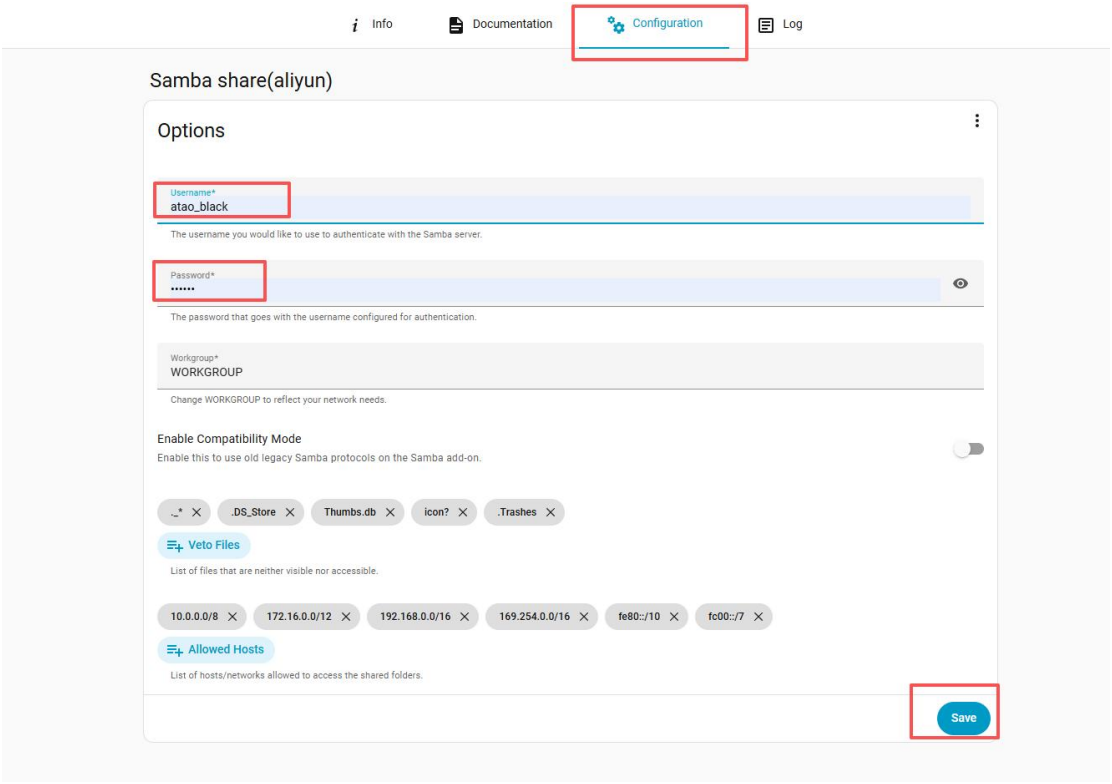
Click INSTALL to install it



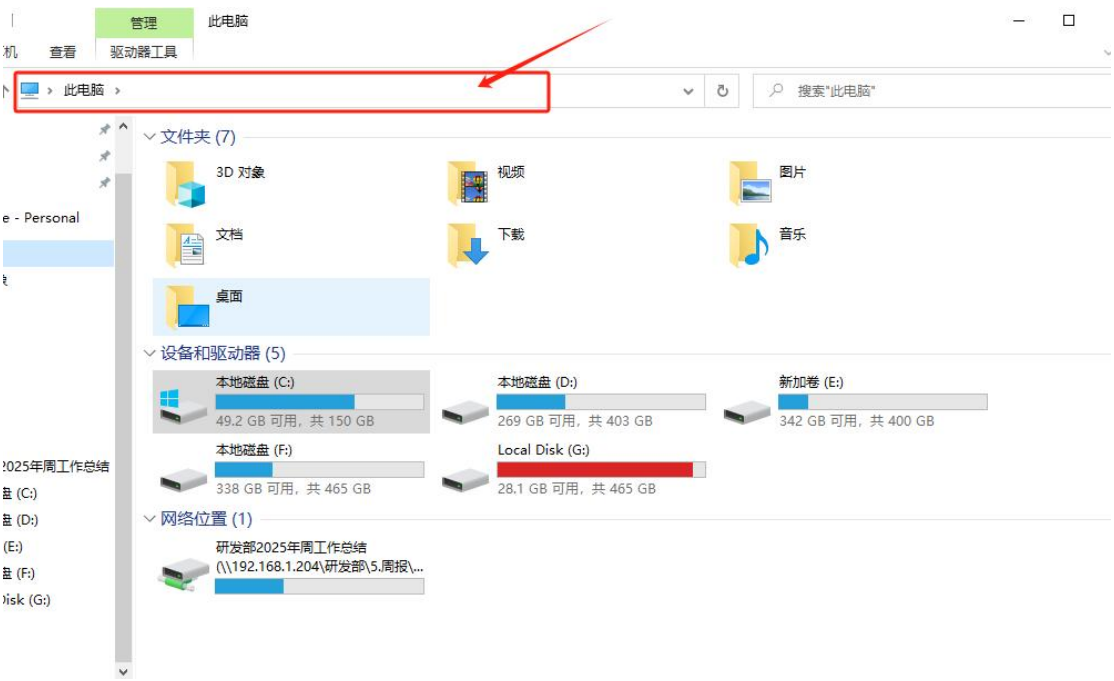
After the installation is completed, remember to start it.



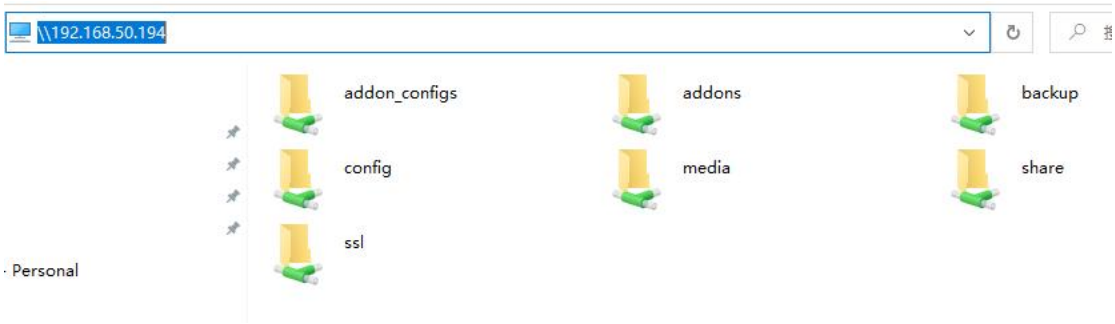
After installation, configure your Samba share with your own username and password—please remember them!



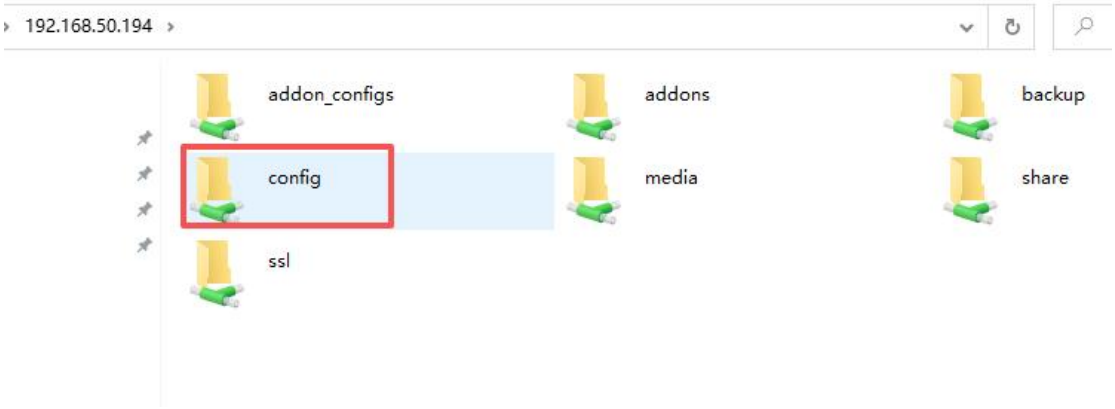
After saving, go to the File Explorer on your computer



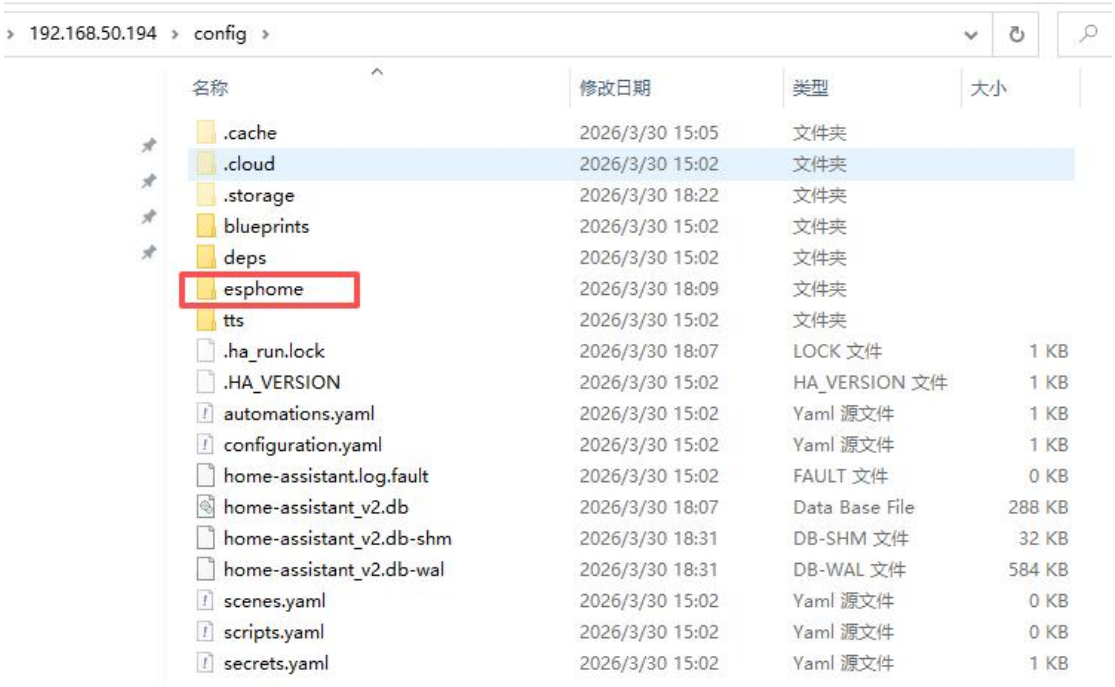
Enter “\\ + your Home Assistant IP address”



Go to config



Then choose esphome



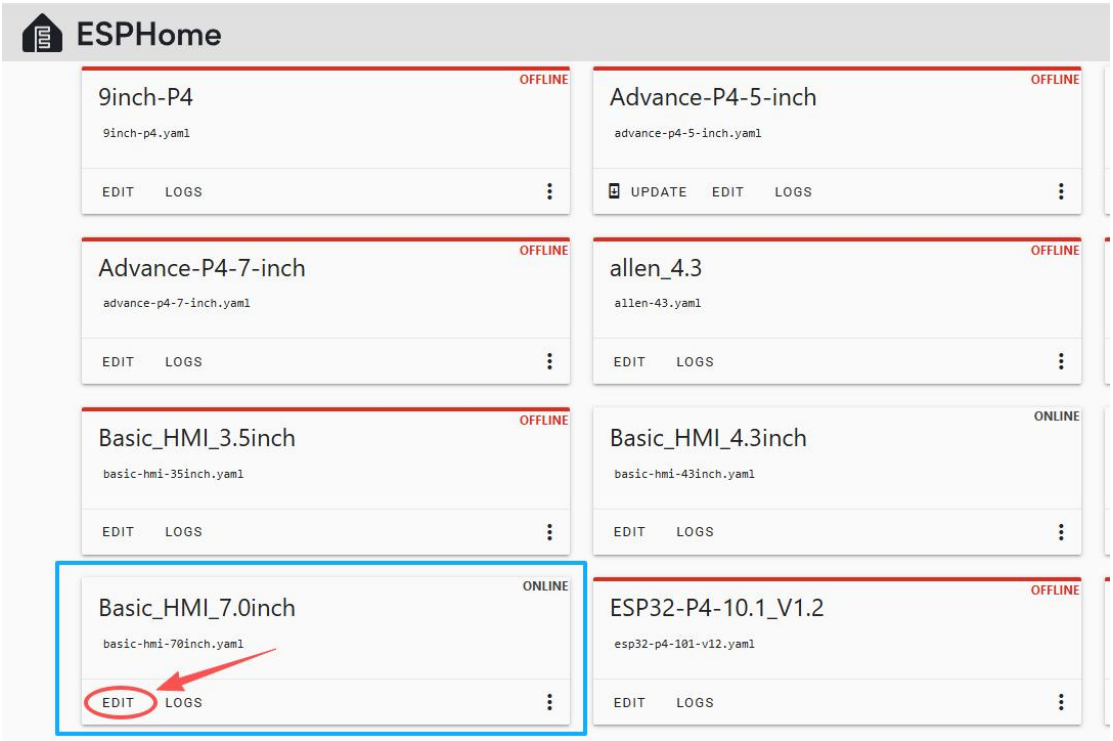
Put the materials you just downloaded into the esphome folder

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

This section defines the base configuration for an ESPHome top-level device, specifying device identification, compilation and build parameters, and high-priority initialization logic upon power-up.

First, it sets the device name and friendly display name, then enables PSRAM support via platformio_options by configuring compilation macros to match the Flash read/write mode required by ESP32-S3 hardware.

Next, it configures a power-on execution script with priority level 600, ensuring it runs before most peripheral components initialize. Using an embedded C++ lambda function, it controls the I2C bus chip PCA9557 (address 0x18) to perform the hardware reset sequence for the GT911 touch controller: first, all PCA9557 pins are configured as outputs, and the reset pin IO0 is pulled low for 20ms to trigger a hard reset of the touch IC; then, IO0 is released by pulling it high, followed by a 100ms delay to allow the touch chip to complete self-check and startup; finally, the register is modified to switch IO1 into input mode, enabling it to receive interrupt signals from the GT911 touch sensor, thereby ensuring proper hardware timing for subsequent touch driver operation.

basic-hmi-70inch.yaml

```yaml
esphome:
  name: basic-hmi-70inch
  friendly_name: Basic_HMI_7.0inch
  platformio_options:                                  # Custom PlatformIO build configuration
    build_flags:
      - "-DBOARD_HAS_PSRAM"                            # Enable PSRAM support in compilation
    board_build.esp-idf.memory_type: qio_opi           # Set memory type for ESP-IDF framework
    board_build.flash_mode: dio                        # Set SPI flash access mode to DIO

  on_boot:                                             # Actions to execute during boot phase
    priority: 600                                      # High priority to run before most components
    then:
      - lambda: |-
          uint8_t addr = 0x18;                         // PCA9557 I2C expander slave address

          uint8_t config[2] = {0x03, 0x00};            // 1. Set all PCA9557 pins as outputs
          id(bus_a).write(addr, config, 2);            // Write configuration via I2C bus

          uint8_t output_low[2] = {0x01, 0x00};        // 2. IO0=LOW, IO1=LOW (Hard reset GT911)
          id(bus_a).write(addr, output_low, 2);        // Apply low signals to reset pins

          delay(20);                                   // 3. Hold reset state for 20 milliseconds

          uint8_t output_high[2] = {0x01, 0x01};       // 4. IO0=HIGH (Release GT911 reset sequence)
          id(bus_a).write(addr, output_high, 2);       // Apply high signal to wake up touch IC

          delay(100);                                  // 5. Wait 100ms for touch IC to initialize

          uint8_t config_input[2] = {0x03, 0x02};      // 6. Set IO1 as input for GT911 interrupt
          id(bus_a).write(addr, config_input, 2);      // Finalize PCA9557 pin configuration
```

2.esp32

This section defines the core low-level hardware configuration for the ESP32 chip. The main controller is selected as the ESP32-S3, paired with the esp32s3box baseboard. It mandates the use of the native ESP-IDF development framework instead of Arduino, and applies batch sdkconfig compilation parameters to achieve specialized performance and memory optimization: maximizing CPU frequency to 240 MHz, expanding data cache by 64 KB to enhance computational speed, globally enabling external PSRAM configured in an 80 MHz eight-line high-speed mode, and activating features that allow program instruction fetching and storage of read-only constant data from PSRAM.

Full support for ESP32-S3-specific PSRAM drivers is enabled, precisely matching the independent PSRAM module settings below. This provides ample high-speed external memory for large-screen LVGL interfaces (800×480), image caching, and touch driver operations, preventing memory overflow, interface lag, or program crashes during intensive screen rendering.

```yaml
esp32:                                                 # ESP32 specific hardware configuration
  board: esp32s3box                                    # Base board model for ESP32-S3
  framework:
    type: esp-idf                                      # Use official ESP-IDF framework, not Arduino
    sdkconfig_options:
      CONFIG_ESP32S3_DEFAULT_CPU_FREQ_240: y           # Set CPU frequency to max (240MHz)
      CONFIG_ESP32S3_DATA_CACHE_64KB: y                # Allocate 64KB for data cache
      CONFIG_SPIRAM_FETCH_INSTRUCTIONS: y              # Allow instruction fetch from SPIRAM
      CONFIG_SPIRAM_RODATA: y                          # Store read-only data in SPIRAM
      CONFIG_SPIRAM_USE: y                             # Enable external SPIRAM globally
      CONFIG_SPIRAM_SPEED_80M: y                       # Set SPIRAM clock speed to 80MHz
      CONFIG_SPIRAM_MODE_OCT: y                        # Use Octal mode for faster RAM access
      CONFIG_ESP32S3_SPIRAM_SUPPORT: y                 # Enable ESP32-S3 specific SPIRAM support
  variant: esp32s3                                     # Specify the chip variant exactly
```

3.psram + logger + api + ota

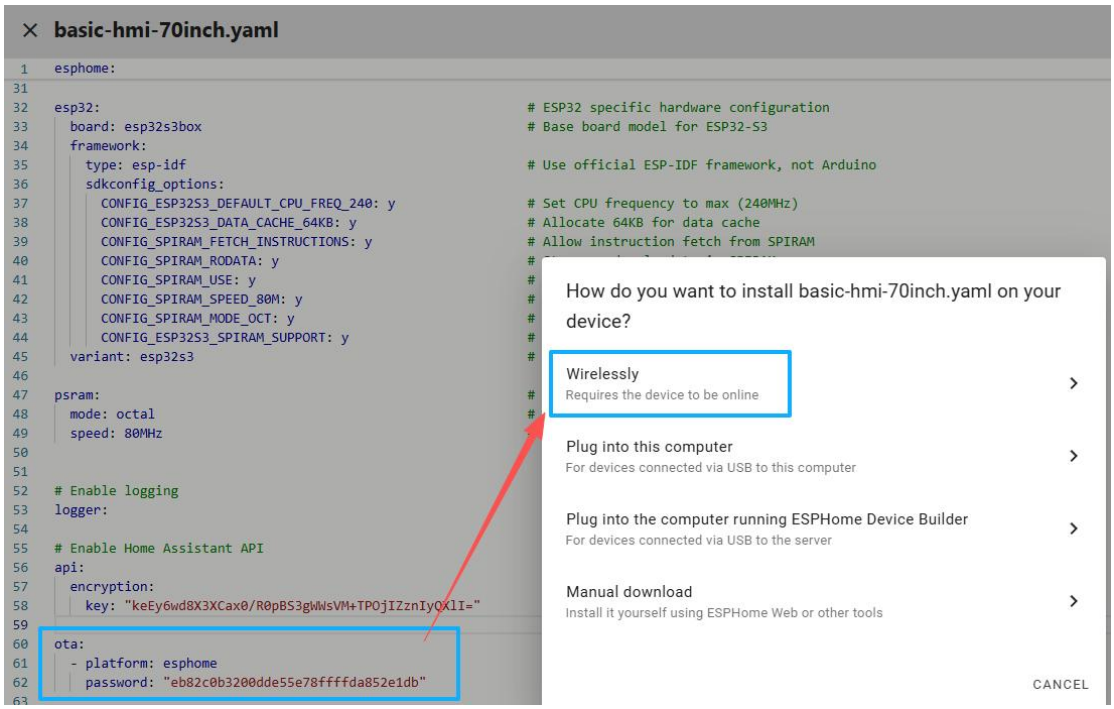
This section first configures the physical parameters for the octal-line PSRAM hardware, specifying an octal SPI mode that matches the motherboard's routing, an 80 MHz operating clock, and ensuring consistency with the relevant SDK configuration parameters in the ESP32 block above to guarantee proper recognition and operation of the external high-speed memory. Subsequently, it enables the system logging component for hardware debugging and troubleshooting, activates the Home Assistant communication API with encryption keys to facilitate data exchange and command transmission between the device and the smart home platform, and configures native OTA wireless firmware updates in ESPHome along with a password for secure access, allowing remote firmware updates without requiring serial cable connections.

```
47  psram:
48    mode: octal
49    speed: 80MHz
50
51
52  # Enable logging
53  logger:
54
55  # Enable Home Assistant API
56  api:
57    encryption:
58      key: "keEy6wd8X3XCax0/R0pBS3gWWsVM+TPOjIZznIyQXlI="
59
60  ota:
61    - platform: esphome
62      password: "eb82c0b3200dde55e78ffffda852e1db"
63
```

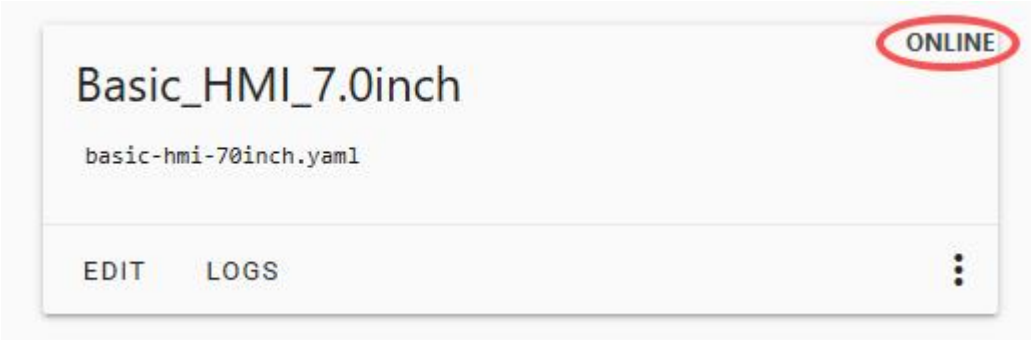
Finally, it's ota:

```
60  ota:
61    - platform: esphome
62      password: "eb82c0b3200dde55e78ffffda852e1db"
```

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.

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

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Basic-Hmi-70Inch"
    password: "sGPi8gC5ATd2"
```

5. External LED

This section is used to create a GPIO switch entity that can be synchronized with Home Assistant. It is bound to the hardware pin GPIO38 through the gpio driver, and the external display entity name is set as "LED Control".
At the same time, an internal identifier "led_switch" is assigned. This allows for remotely toggling the pin level and controlling the external LED hardware in the smart home platform. It can also be called by this internal ID in the logic of the local LVGL interface, touch events, etc., to achieve bidirectional interaction between local touch control and remote control.

```
switch:
  - platform: gpio
    pin: 38
    name: "LED Control"
    id: led_switch
```

6. Screen backlight adjustment

This section is divided into two parts: hardware PWM output and the physical light entity. First, the ledc hardware PWM driver is used to bind GPIO2 as the control pin for the screen backlight, setting a 1220Hz low-frequency PWM to eliminate visible flicker and assigning the internal identifier gpio_backlight_pwm.

Then, based on the monochrome light component, the aforementioned PWM output is associated to generate a HA light entity named Display Backlight with an internal ID of back_light, which supports brightness adjustment. At the same time, the power-on recovery mode is configured as ALWAYS_ON to ensure that the screen backlight automatically lights up after the device is powered off and restarted. This can be uniformly controlled in local automation logic and the Home Assistant platform for the screen brightness.

```
output:
  - platform: ledc
    pin: 2
    frequency: 1220
    id: gpio_backlight_pwm

light:
  - platform: monochromatic
    output: gpio_backlight_pwm
    name: Display Backlight
    id: back_light
    restore_mode: ALWAYS_ON
```

7.DHT20

This section is divided into two main parts: I2C bus hardware configuration and AHT20 temperature/humidity sensor driver. It fully supports the underlying infrastructure for the shared communication bus of the touch expansion chip and the temperature/humidity sensor: Firstly, configure the hardware I2C physical channel, designate the data pin SDA as GPIO19, the clock pin SCL as GPIO20, and enable the parameter "scan: true" to allow the device to automatically traverse the entire bus and print the addresses of all attached devices upon power-on, facilitating troubleshooting of I2C device wiring and address conflicts during the development stage.

At the same time, assign a unique internal identifier "bus_a" to the bus as the shared communication channel for the subsequent PCA9557, GT911, and AHT20; subsequently, connect the AHT20 digital temperature/humidity sensor, and select a universal driver compatible with the AHT10 series to bind the aforementioned I2C bus ID "bus_a", fill in the standard I2C address of the chip 0x38, and clearly specify the device model "variant" as AHT20 to adapt to the internal data parsing logic of the sensor. Split the collected temperature and humidity data into two paths for output, respectively setting internal IDs "temperature70" and "humidity70" exclusively for the LVGL graphical interface to render and display the values in real time, and generate an external entity name to synchronize with Home Assistant, setting a 5-second interval for collecting sensor data.

```
i2c:
  sda: 19
  scl: 20
  scan: true
  id: bus_a

sensor:
  - platform: aht10
    i2c_id: bus_a
    address: 0x38
    variant: AHT20
    temperature:
      id: temperature70
      name: "HMI-70 Temperature"
    humidity:
      id: humidity70
      name: "HMI-70 Humidity"
    update_interval: 5s
```

8. touch screen

This section is divided into two parts: the configuration of the GT911 capacitor touch driver and the virtual touch buttons on the screen. It relies on the I2C bus "bus_a" defined in the previous text to implement the complete touch function link: Firstly, load the dedicated driver for the Hikvision GT911 touch IC, allocate the internal identifier "my_touchscreen" to connect with the LVGL graphics library to receive touch coordinate signals, bind to the common I2C bus "bus_a", associate the 800×480 main display "main_display" to complete the coordinate mapping, set the touch point position to be polled once every 20 milliseconds to balance the touch response speed and system resource consumption, and turn off all coordinate axis transformations without flipping or swapping XY to ensure that the touch point position is completely corresponding to the screen position;

Then, based on this touch device, create a virtual binary sensor for a rectangular touch area, naming it "Light Touch Button", define the rectangular area on the screen with X-axis from 150 to 202 pixels and Y-axis from 300 to 352 pixels as the touch button, configure the pressing trigger logic, and once the user clicks on this screen area, automatically flip the internal ID "led_switch" of the GPIO indicator light switch state.

```
touchscreen:
  platform: gt911
  id: my_touchscreen
  i2c_id: bus_a
  display: main_display
  update_interval: 20ms
  transform:
    mirror_x: false
    mirror_y: false
    swap_xy: false

binary_sensor:
  - platform: touchscreen
    name: "Light Touch Button"
    touchscreen_id: my_touchscreen
    x_min: 150
    x_max: 202
    y_min: 300
    y_max: 352
    on_press:
      then:
        - switch.toggle: led_switch
```

9.display

This section includes two configuration modules: the RGB parallel DPI display hardware driver and the network time synchronization module. The former uses the ESP-IDF-specific rpi_dpi_rgb parallel color screen driver, assigns the internal display ID as main_display, sets the pixel color arrangement as RGB, does not reverse the screen color, disables the built-in periodic refresh and automatic screen clearing functions, and delegates the unified control of screen rendering by LVGL to avoid tearing and ghosting. The physical resolution of the screen is fixed at 800×480. The

GPIO corresponding to the DPI timing control pins DE, HSYNC, VSYNC, and PCLK are fully defined, with the pixel clock frequency locked at 15 MHz to match the screen specifications. The parameters for the front shoulder, pulse width, and rear shoulder timing of the horizontal and vertical synchronization are filled in strictly according to the screen manual to ensure normal screen output.

At the same time, the 5 red, 6 green, and 5 blue RGB565 parallel data pins are fully listed to build a complete hardware display path. The latter enables the SNTP network time synchronization component and assigns an internal ID of time_comp. It can pull the network standard time and provide a standard time reference for the automated logic of displaying the clock, automatic control of backlight, etc. in the LVGL interface.

```yaml
display:
  - platform: rpi_dpi_rgb
    id: main_display

    color_order: RGB
    invert_colors: false

    update_interval: never
    auto_clear_enabled: false

    dimensions:
      width: 800
      height: 480

    de_pin: 41
    hsync_pin: 39
    vsync_pin: 40
    pclk_pin: 0

    pclk_frequency: 15MHz

    hsync_front_porch: 40
    hsync_pulse_width: 48
    hsync_back_porch: 40

    vsync_front_porch: 1
    vsync_pulse_width: 31
    vsync_back_porch: 13

    data_pins:
      red: [14, 21, 47, 48, 45]
      green: [9, 46, 3, 8, 16, 1]
      blue: [15, 5, 6, 7, 4]

time:
  - platform: sntp
    id: time_comp
```

10. lvgl

This section is the overall configuration of the LVGL graphical interface. Before binding, the main display with a resolution of 800×480 and the touch device my_touchscreen defined earlier are used as the rendering carrier and touch input source for the interface. The log output level of LVGL is set to INFO for easier debugging.

The color depth is 16-bit RGB565, which is consistent with the screen and image file formats to reduce color conversion costs. The global interface background is set to pure white (0xFFFFFF),

and the default basic font is unscii_8. In the widgets block, all fixed-coordinate UI controls are statically defined: Place the logo image labeled lvgl_logo on the screen (300,40), the light icon lvgl_light is positioned at (150,300) and completely overlaps with the touch button area defined earlier.

The temperature and humidity icons lvgl_temp and lvgl_hum are respectively placed at (390,280) and (630,280). At the same time, three text labels using the large Montserrat_24 font are created. The led_status defaults to OFF to display the status of the light switch, and the tem and hum labels initially have no decimal reading and read the real-time values of the temperature and humidity collected by AHT20.

The three text labels are uniformly arranged under the corresponding icons for intuitive comparison. All controls are assigned independent internal IDs for subsequent dynamic switching of image materials and refreshing of text values, achieving real-time linkage between the interface state and hardware sensors, switches and devices.

```yaml
lvgl:                                          # LVGL (Graphics Library) UI layout
  displays:
    - main_display                             # Target display for UI rendering
  touchscreens:
    - my_touchscreen                           # Source for UI touch interactions
  log_level: INFO                              # Set LVGL internal logging verbosity
  color_depth: 16                              # Match image color depth (RGB565)
  bg_color: 0xFFFFFF                           # Set global background color to white
  text_font: unscii_8                          # Default font used if not specified

  widgets:                                     # UI element definitions
    - image:
        id: lvgl_logo                          # Internal ID for the logo widget
        src: small_logo                        # Reference to loaded small_logo asset
        x: 300                                 # X-axis position on screen (px)
        y: 40                                  # Y-axis position on screen (px)
    - image:
        id: lvgl_light                         # Internal ID for the light icon widget
        src: no_light                          # Reference to loaded on_light asset
        x: 150                                 # Match X position with touch zone
        y: 300                                 # Match Y position with touch zone
    - image:
        id: lvgl_temp                          # Internal ID for the temperature icon widget
        src: small_temp                        # Reference to loaded temperature asset
        x: 390                                 # X-axis position on screen (px)
        y: 280                                 # Y-axis position on screen (px)
    - image:
        id: lvgl_hum                           # Internal ID for the humidity icon widget
        src: small_hum                         # Reference to loaded humidity asset
        x: 630                                 # X-axis position on screen (px)
        y: 280                                 # Y-axis position on screen (px)

    - label:
        id: led_status                         # Internal ID for LED text status
        text_font: montserrat_24               # Use larger, smoother Montserrat font
        text: "OFF"                            # Default starting text
        x: 160                                 # X-axis position near the light icon
        y: 400                                 # Y-axis position below the light icon
    - label:
        id: tem                                # Internal ID for temperature text value
        text_font: montserrat_24               # Use larger, smoother Montserrat font
        text:
          format: "%.0f"                       # Display value with no decimal places
          args: ['id(temperature70).state']    # Fetch live data from AHT20 sensor
        x: 400                                 # X-axis position near temperature icon
        y: 400                                 # Y-axis position below temperature icon
```

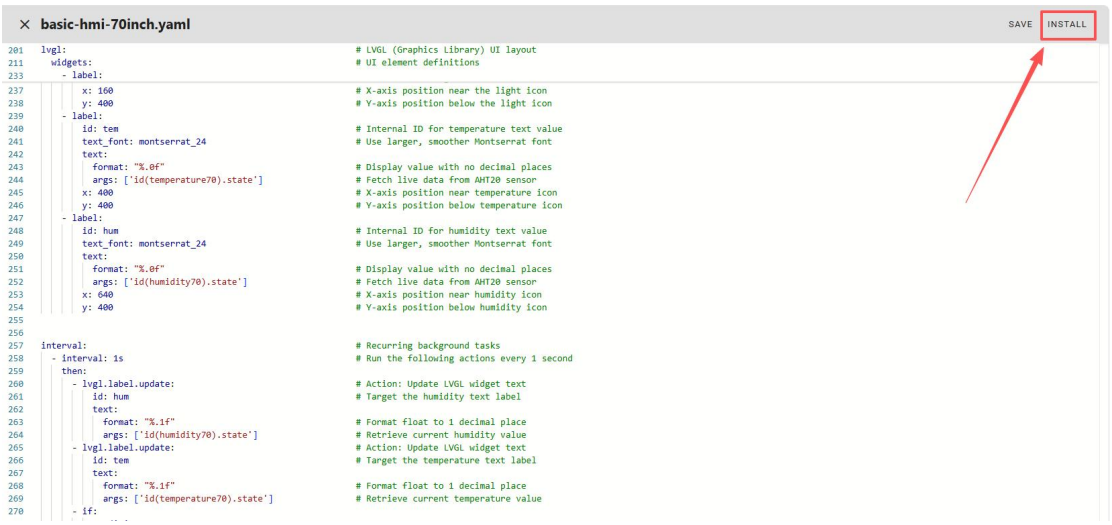
11. interval

This section has a background loop task with a configuration period of 1 second. Every second, it automatically executes the entire UI dynamic refresh logic: Firstly, it calls the LVGL tag update instructions to read the real-time values of the temperature and humidity variables temperature70 and humidity70 output by the AHT20 sensor, formatting them with one decimal place, and refreshes the interface text controls "hum" and "tem" in this format, allowing the screen to continuously display the latest environmental data; then, it adds a conditional judgment to read the on/off state of the GPIO switch led_switch. If the switch is in the on state, it switches the interface light icon lvgl_light to the on style on_light, and modifies the status text led_status to ON; if the switch is off, it switches to the off icon no_light and updates the text to OFF. This achieves the effect of real-time refresh of the screen temperature and humidity values, and the synchronization of the light icon and status text with the hardware switch state.

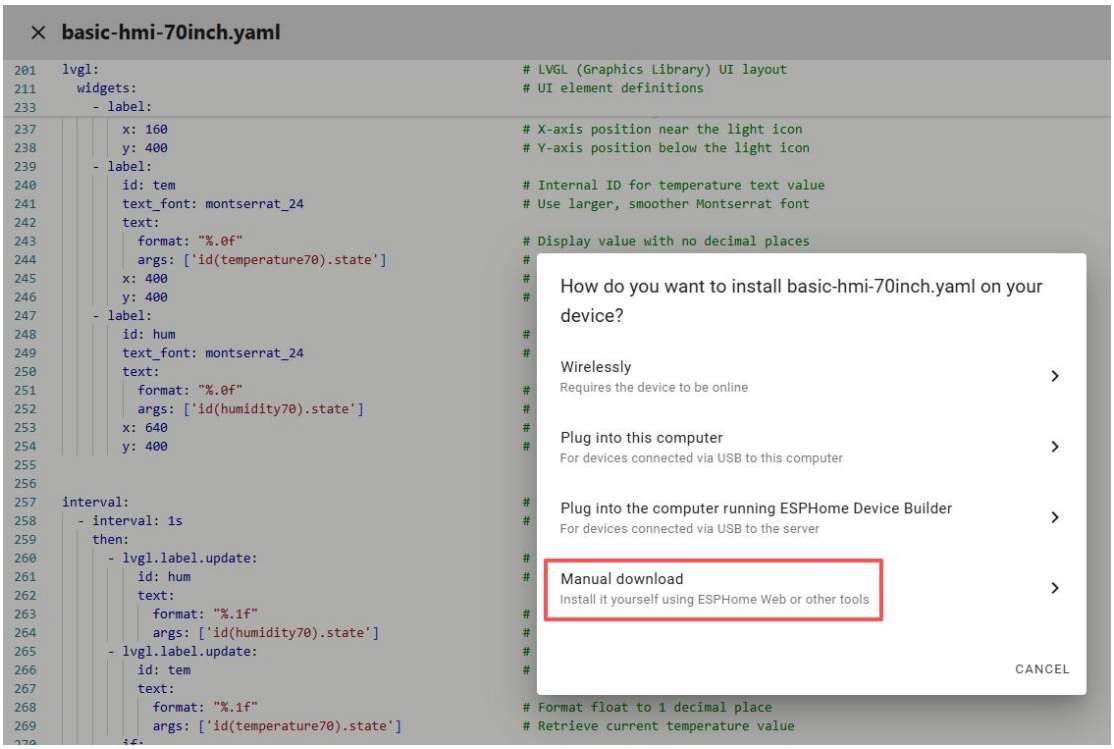
```
interval:                                          # Recurring background tasks
  - interval: 1s                                   # Run the following actions every 1 second
    then:
      - lvgl.label.update:                         # Action: Update LVGL widget text
          id: hum                                  # Target the humidity text label
          text:
            format: "%.1f"                         # Format float to 1 decimal place
            args: ['id(humidity70).state']         # Retrieve current humidity value
      - lvgl.label.update:                         # Action: Update LVGL widget text
          id: tem                                  # Target the temperature text label
          text:
            format: "%.1f"                         # Format float to 1 decimal place
            args: ['id(temperature70).state']      # Retrieve current temperature value
      - if:
          condition:
            switch.is_on: led_switch
          then:
            - lvgl.image.update:
                id: lvgl_light
                src: on_light

            - lvgl.label.update:
                id: led_status
                text: "ON"

          else:
            - lvgl.image.update:
                id: lvgl_light
                src: no_light

            - lvgl.label.update:
                id: led_status
                text: "OFF"
```

12.Upload the complete code

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

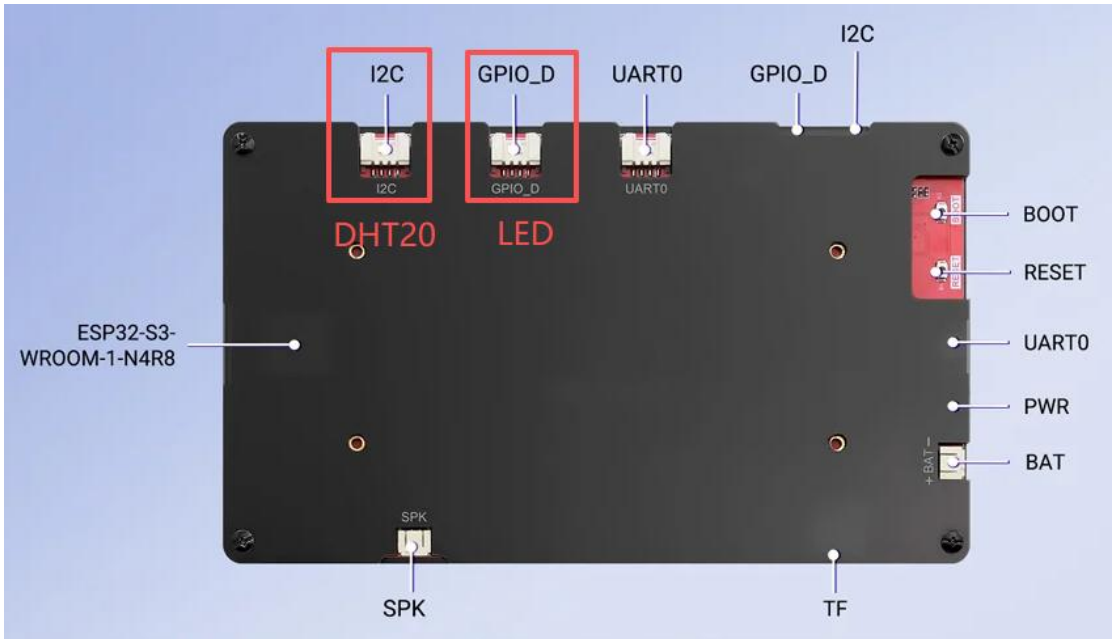


Then select "Manual download"



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.

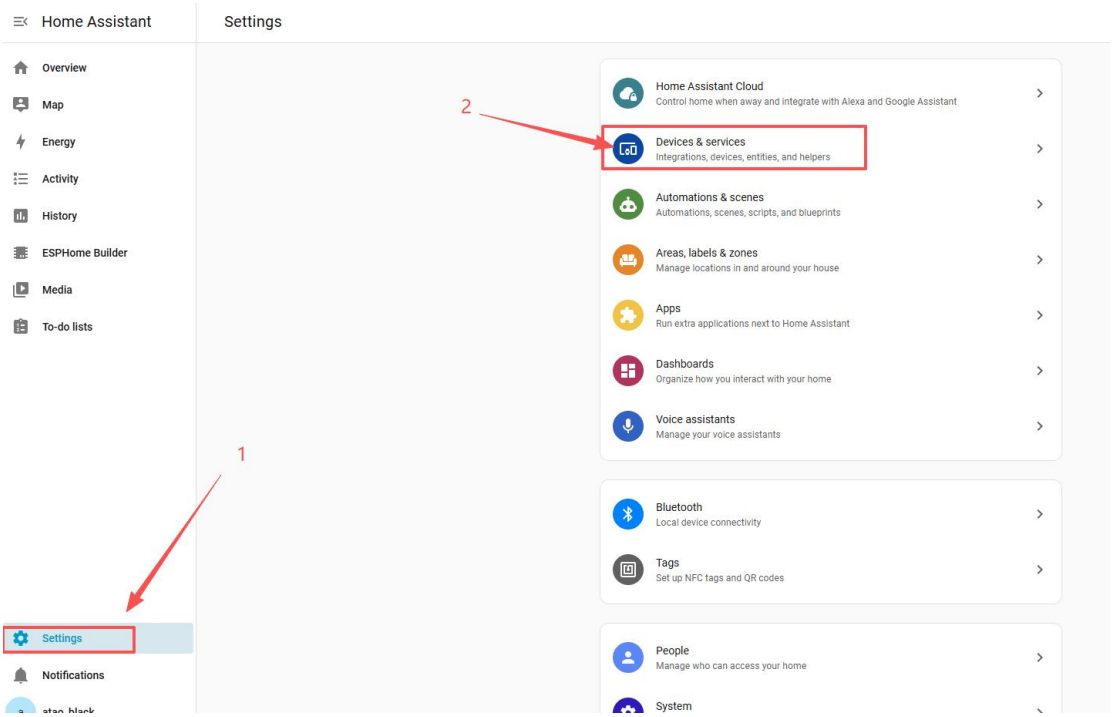


Then you will be able to see the real-time display of temperature and humidity data on the screen.

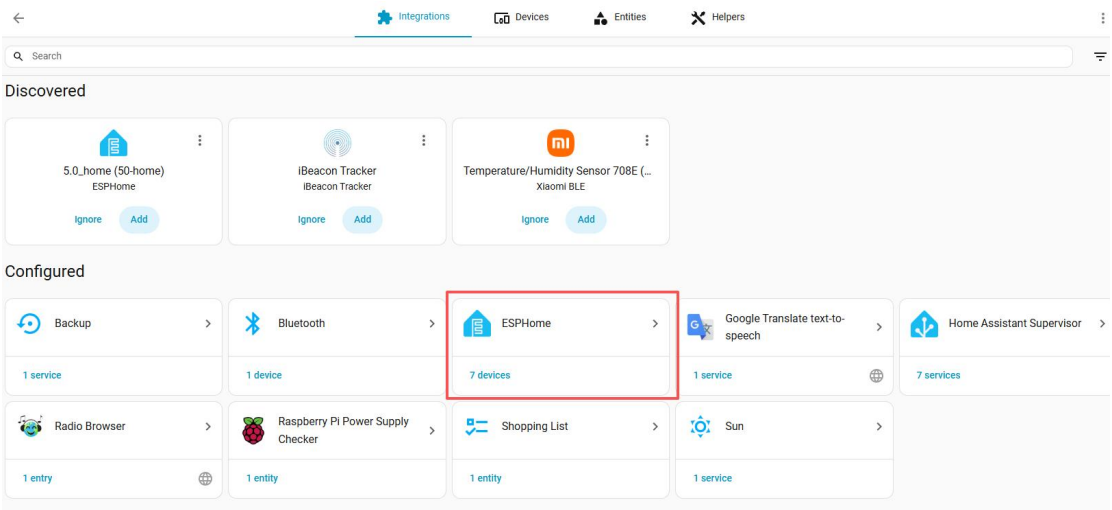
13.Remote observation and control

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.

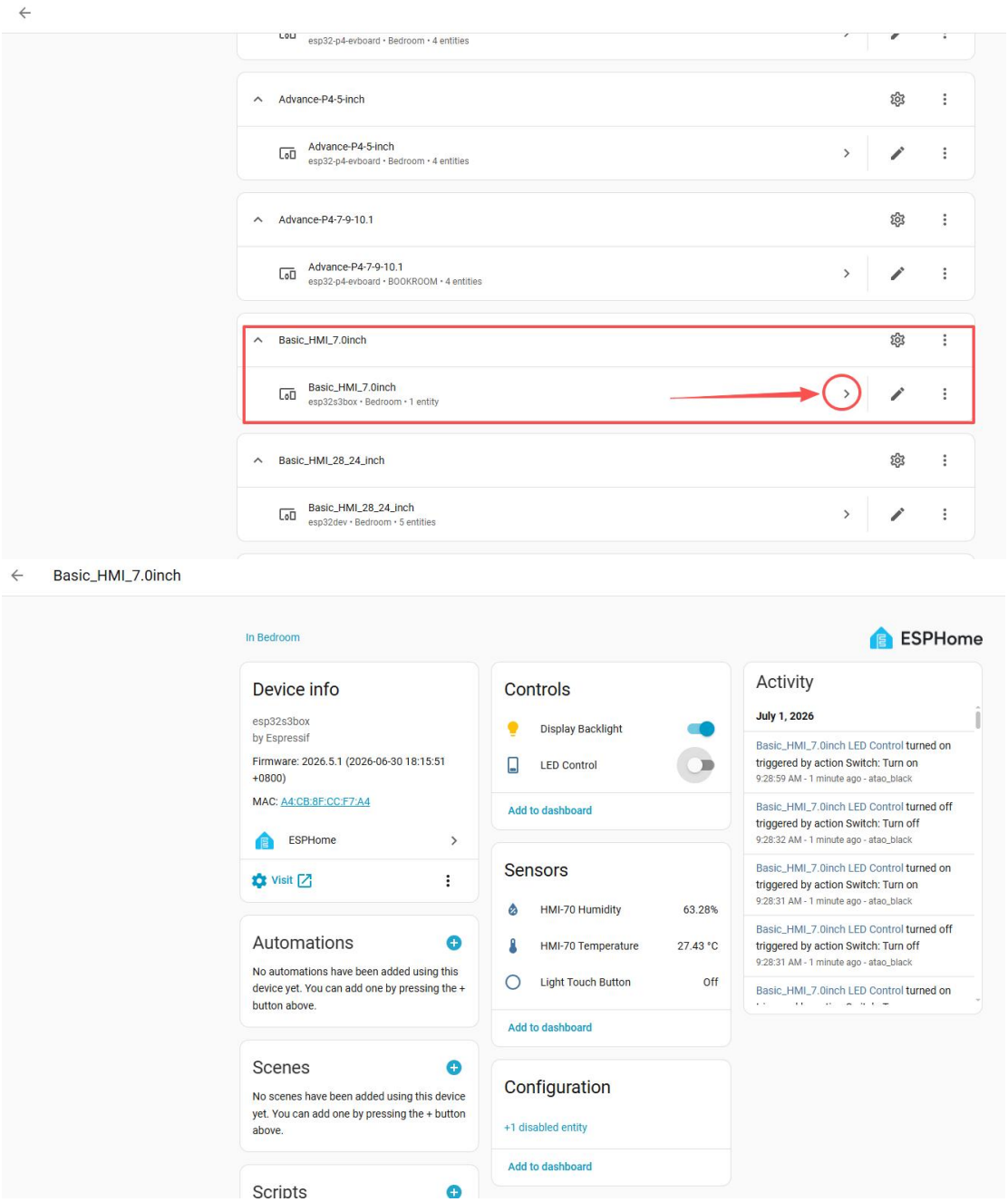
go to settings and select devices



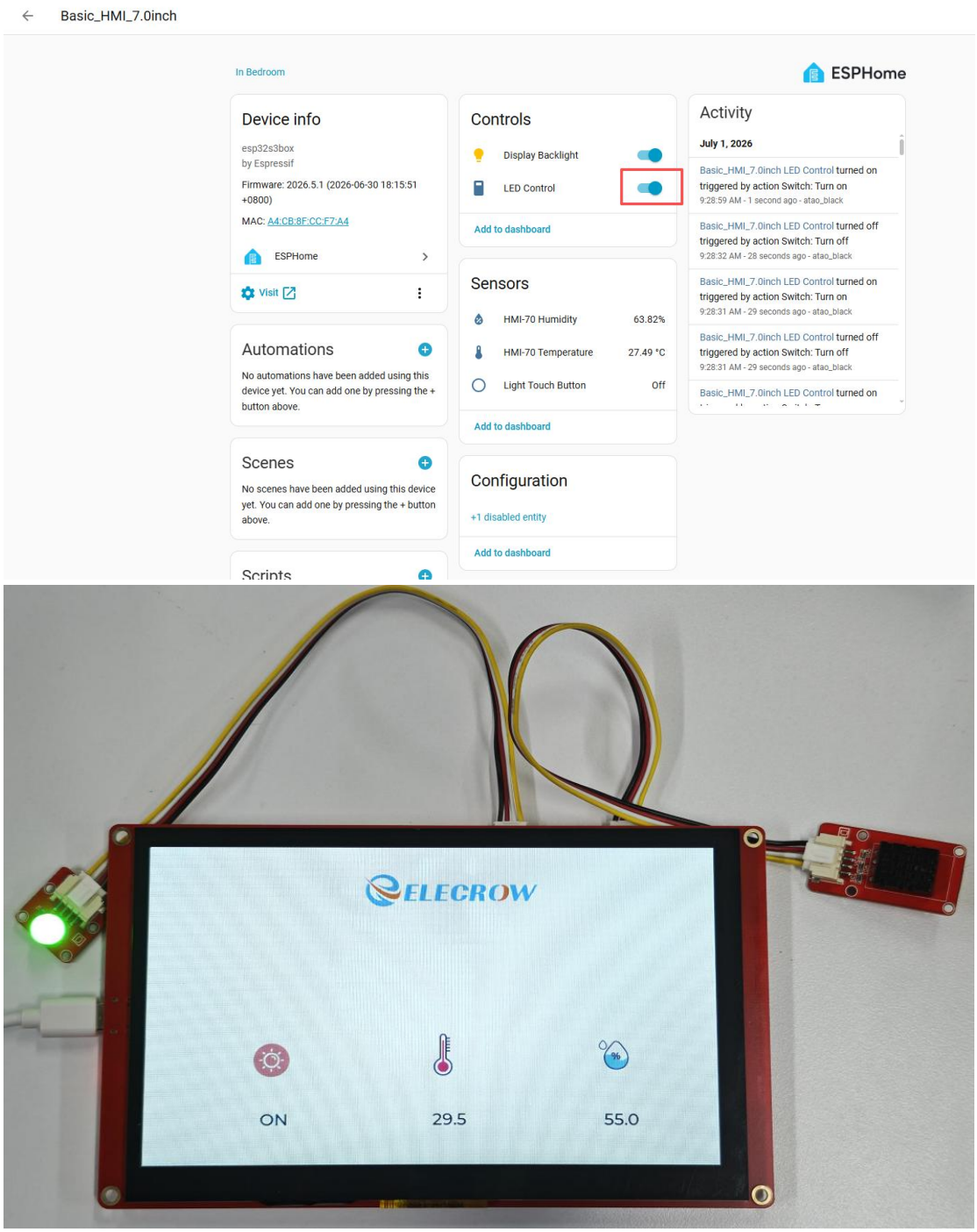
Choose ESPHome



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

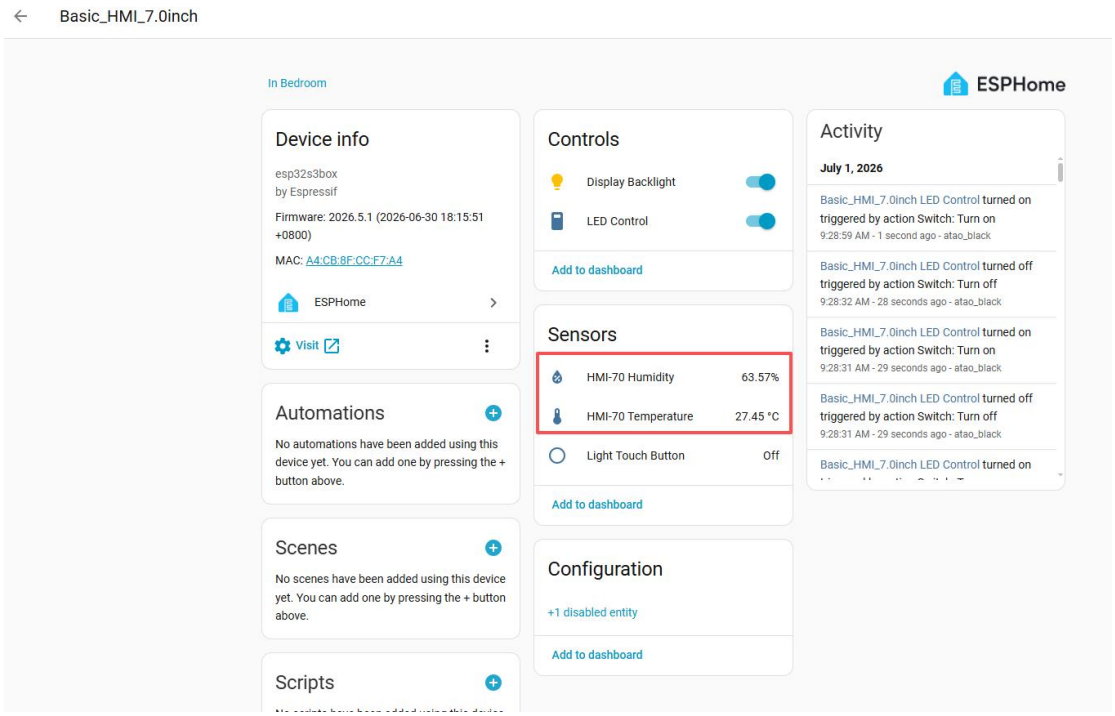


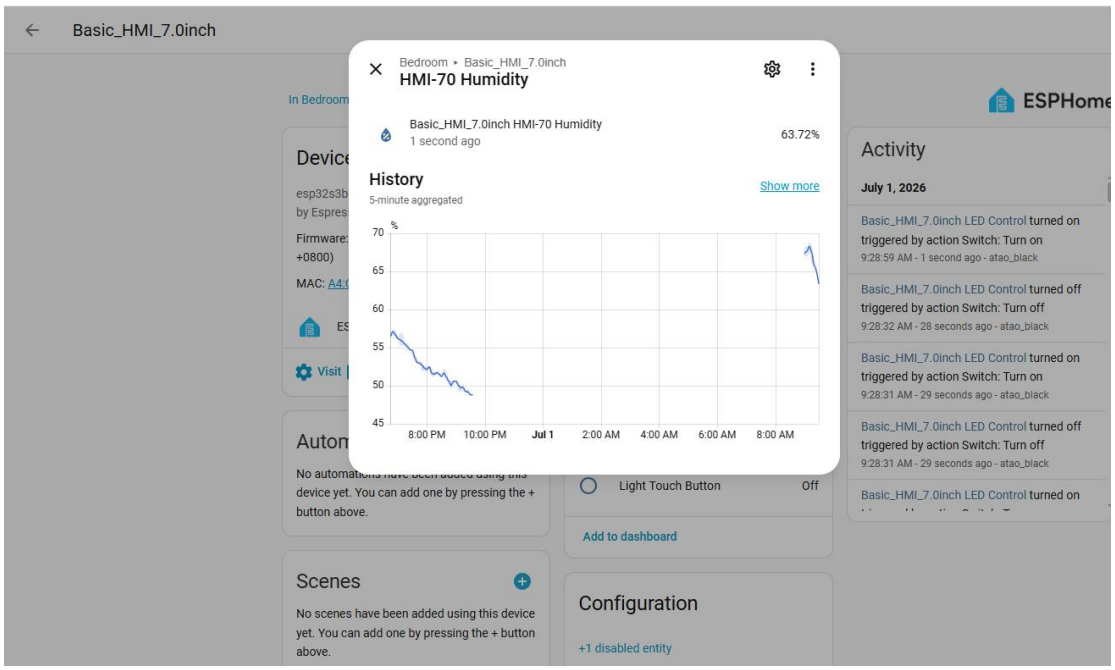
Next, when you click the switch of the LED, you will be able to see that the indicator light on the screen turns on, and the feedback from the LED is also quite obvious.

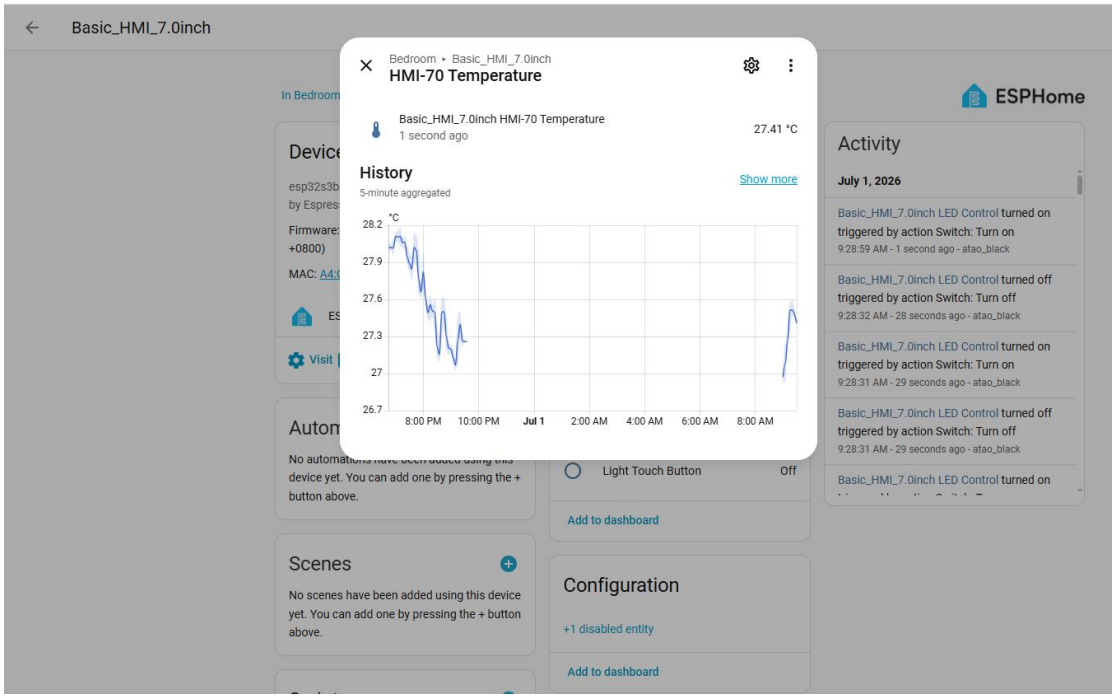


Next, when you click the switch of the LED, you will be able to see that the display light on the screen turns off, and the feedback from the LED is also quite obvious.

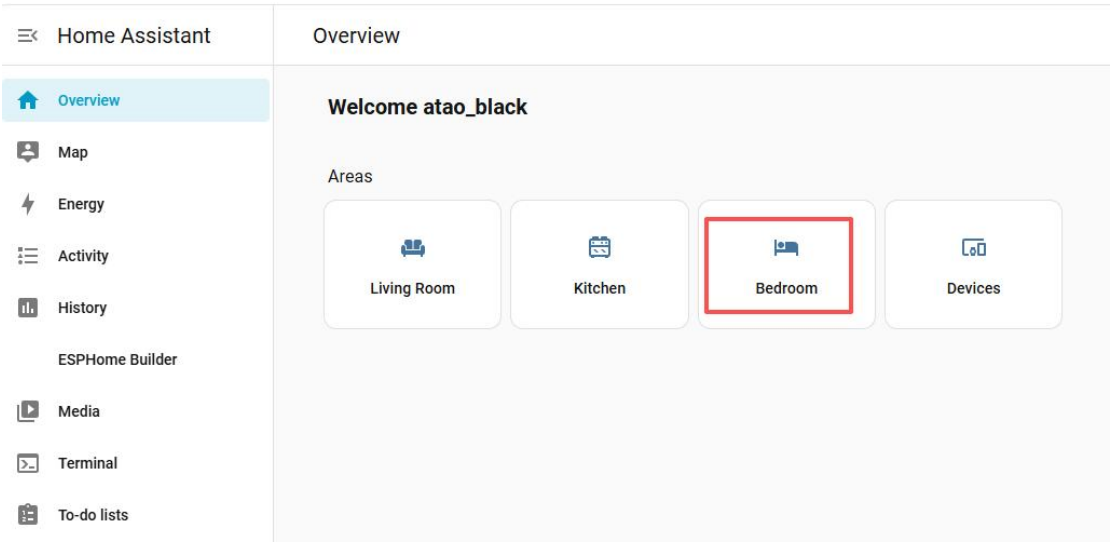
And you can also see the historical data of temperature and humidity collection from here.







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.