

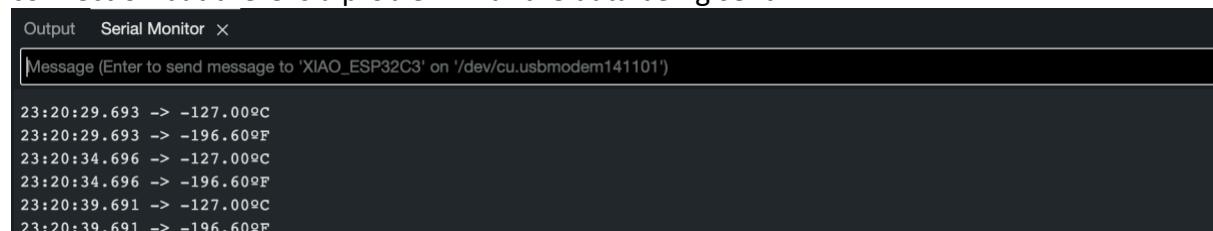
Internet of Things logbook – DJ306 Smart Cooking Pot Thermometer

After deciding to make a Smart cooking pot thermometer I had to find the appropriate sensors in order to implement a device that can fit that need. The DHT11 sensor detects heat and humidity however it wouldn't be suitable as it cannot be submerged in water luckily there is a submergible heat detective sensor being the DS18B20 and I checked the supported temperatures are -55 – 125 degrees Celsius which is perfect for this device as the boiling point of water is 100 degrees meaning it can reach that heat without being damaged.

20th November

I found the library for the sensor which enabled me to use it as a starting point for the code and understanding it a bit better. When setting up the DS18B20 sensor at first, I was getting this at the start I think this has to be due to how the wiring is currently setup or the data pin cable is loose.

I researched and found out -127 degrees is a placeholder value for when there is a connection but there is a problem with the data being sent.

A screenshot of a Serial Monitor window. The title bar says "Output Serial Monitor x". Below the title bar is a text input field with the placeholder text "Message (Enter to send message to 'XIAO_ESP32C3' on '/dev/cu.usbmodem141101')". The main area of the window displays a series of temperature readings in a monospaced font. Each line shows a timestamp, a time difference, and a temperature value in both Celsius and Fahrenheit.

```
23:20:29.693 -> -127.00°C
23:20:29.693 -> -196.60°F
23:20:34.696 -> -127.00°C
23:20:34.696 -> -196.60°F
23:20:39.691 -> -127.00°C
23:20:39.691 -> -196.60°F
```

This was due to the data pin being connected to the wrong pin as on the board the pin inputs at the bottom were positioned one to the left and after successfully declaring the correct pin the sensor started reading the correct temperature from the sensor.

I initially wanted to have a water flow sensor too however this was a lot more difficult than I expected as unlike for the DS18B20 sensor there wasn't a library I could find so at this stage it is looking unlikely that I will be implementing it. However, looking at it from a bigger picture I think the DS18B20 will enable me to cover all the crucial features I want and make the device functioning.

After successfully declaring the right pin, the sensor was successfully displaying the correct temperature and below is the code I used to display it assigning variables to display the temperature alongside including the correct degrees type after the temperature.

```

20 void loop() {
21   sensors.requestTemperatures();
22   float temperatureC = sensors.getTempCByIndex(0);
23   float temperatureF = sensors.getTempFByIndex(0);
24   Serial.print(temperatureC);
25   Serial.println("°C");
26   Serial.print(temperatureF);
27   Serial.println("°F");
28   delay(5000);
29 }

```

Output Serial Monitor X

Message (Enter to send message to 'XIAO_ESP32C3' on '/dev/cu.usbmodem142101')

```

23:36:51.716 -> 23.69°C
23:36:51.716 -> 74.64°F
23:36:57.262 -> 23.62°C
23:36:57.262 -> 74.52°F
23:37:02.788 -> 23.62°C
23:37:02.788 -> 74.52°F

```

In this early stage I am currently developing the sensor to connect to WIFI and be able to send through the data so it can be displayed on another device, I plan on doing this by using HTML to make a website and connecting the

board to WIFI.

```

1  #include <WiFi.h>
2  #include <WebServer.h>
3  #include <DallasTemperature.h>
4  |
5  const char* ssid = "SHELL-483758";
6  const char* password = "hkfKvfM3MQhk";

```

I imported the WIFI library needed to enable connection to my home WIFI by adding the SSID and the password so that upon running a connection can be made.

```

void setup() {
  Serial.begin(115200);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(5000);
    Serial.println(".");
  }
}

```

Here I added the code to connect to the WIFI.

```

Serial.println();
Serial.println("Connected to WiFi");

```

Upon connecting the program will constantly print full stops until it connects, and the while statement ends it moves on to print a message letting the user know it connected WIFI and below is the rest of my code for connecting and also displaying the temperature on the website.

```

30 sensors.begin();
31
32 server.on("/", []() {
33     sensors.requestTemperatures();
34     float temperatureC = sensors.getTempCByIndex(0);
35     float temperatureF = sensors.getTempFByIndex(0);
36
37     String webpage = "<html><body>";
38     webpage += "<h1>Temperature Monitoring</h1>";
39     webpage += "<p>Temperature in Celsius: " + String(temperatureC) + " &#8451;</p>";
40     webpage += "<p>Temperature in Fahrenheit: " + String(temperatureF) + " &#8457;</p>";
41     webpage += "</body></html>";
42
43     server.send(200, "text/html", webpage);
44 });
45
46 server.begin();
47
48
49 void loop() {
50     server.handleClient();
51 }

```

However, when running the code despite making a connection through Arduino I think despite being a proper IDE it can't load the website that is being generated in a test environment so I think this could be either making the file somewhere on my laptop.

23rd November

```

25 Serial.println();
26 Serial.println("Connected to WiFi");
27 Serial.print("IP address: ");
28 Serial.println(WiFi.localIP());

```

I had to research this issue as I wasn't too familiar with using WIFI and Arduino code. I found a good guide explaining that

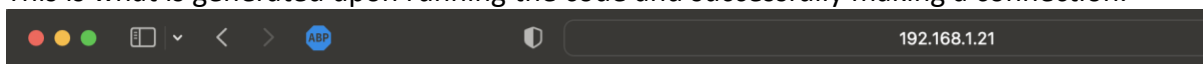
when making a connection to WIFI a local IP is generated so after printing the message letting the user know they have connected I added the correct code for displaying the IP that is generated in order to access the website that is being made upon running the code.

```

25 Serial.println();
26 Serial.println("Connected to WiFi");
27 Serial.println(WiFi.localIP());

```

This is what is generated upon running the code and successfully making a connection.



Temperature Monitoring

Temperature in Celsius: 26.12 °C

Temperature in Fahrenheit: 79.02 °F

When pasting the generated IP into my browser the code I had to generate the temperature alongside the data being read by the sensor worked as upon refreshing the current temperature would be displayed however for a smart device this would not be a viable option as the user would have to constantly refresh the website to get the current temperature and there could be a sudden change with the user may not pick up on and this does not alert the user of anything however this is a prototype of the type of connection I would want there to be with the board.

9th December

The next phase is transitioning to Arduino Cloud, I started off by setting up all the required devices and things needed for this to work.



Here I added my board and correctly declared all the settings for it.

Cloud Variables

ADD

Name ↓	Last Value	Last Update
<input type="checkbox"/> Message String message;		12 Dec 2023 16:55:44
<input type="checkbox"/> Temperature CloudTemperatureSensor temperatur_	27.813	12 Dec 2023 17:41:59

Associated Device

XiaoEsp32

ID: 8c46d631-4a82-4905-8a84-...

Type: XIAO_ESP32C3

Status: Offline

Change Detach

Network

Wi-Fi Name: SHELL-4...

Password:

Secret Key:

Change

I added two cloud variables, one for the message to generate appropriate messages based on inputs etc and one for the temperature. My original code wasn't working, and I was getting error messages saying the previous libraries I was using were not recognised by the system and this is because it is a completely separate IDE so I imported the same libraries on the cloud IDE

```
1
2 // OneWire - Version: Latest
3 #include <OneWire.h>
4 // DallasTemperature - Version: Latest
5 #include <DallasTemperature.h>
6 // GPIO where the DS18B20 is connected to
7 #define ONE_WIRE_BUS D8
8 OneWire oneWire(ONE_WIRE_BUS);
9 DallasTemperature sensors(&oneWire);
16 The following variables are automatically generated and updated when changes are made to the Thing
17
18 String message;
19 CloudTemperatureSensor temperature;
```

At the top of the code, I realised that the cloud variables I set up are automatically generated upon running and I don't need to redeclare them. I also think this means I should be able to call the variables anywhere within the code.

```

51 void loop() {
52   ArduinoCloud.update();
53   //request temperature to all devices on data line
54   sensors.requestTemperatures();
55
56   Serial.print("Celsius temperature: ");
57   Serial.print(sensors.getTempCByIndex(0));
58
59   Serial.print(" - Fahrenheit temperature: ");
60
61   Serial.println(sensors.getTempFByIndex(0));
62
63   temperature=sensors.getTempCByIndex(0);
64
65   delay(1000);
66
67 }

```

Here is my code for displaying the WIFI but also getting reading from the sensor by using the request.Temperatures call. As the temperature variable is a cloud variable, I can just call it anywhere from the code and by using the call method to get the temperature I assigned the temperature being read to the variable.

Despite the code all making sense I was still getting an error that with my board and that it wasn't able to make a connection despite the board being plugged in and every field being correctly declared on the device settings. I think this has to be to do with establishing a connection to the board however the right port isn't showing up on the website which makes sense as the website won't have access to the ports so I think there might be some process I will need to do in order to enable this.

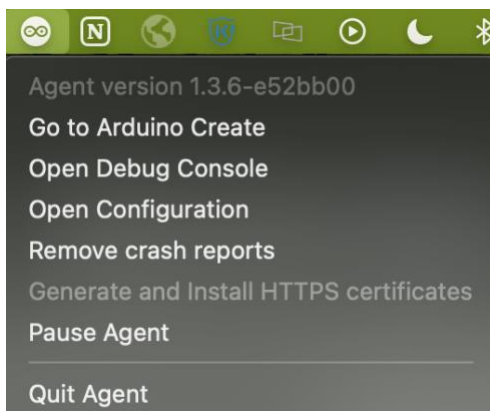
Executing command: exit status 2

```

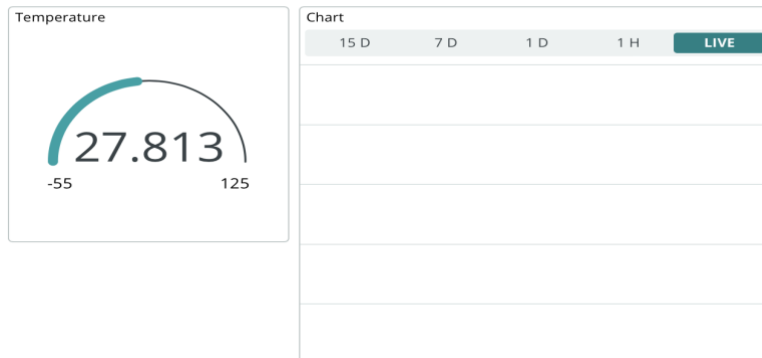
esptool.py v4.2.1
Serial port /dev/cu.usbmodem142101
A fatal error occurred: Could not open /dev/cu.usbmodem142101, the port doesn't exist

```

After researching this issue I realised it was a Mac issue and without a specific driver called the Arduino create agent the board was not found. After installing the agent and installing the HTTPS certificates I was finally able to establish a connection to my board and it was finally displaying my device



The next step is making a dashboard so that I can display the data coming in from the sensor. Arduino has built in widgets that can be used alongside the cloud variables. I added two charts for displaying the temperature and I assigned the temperature variable that I have previously declared, and the temperature was showing and it updates in real time as the board is run. Arduino has an app for which it can be tracked and the same dashboard I had on the app correctly displayed the temperature after assigning the cloud variable for temperature.



Now that this was setup the next stage was setting up alerts however this will not be feasible after I realised that in order to implement events you need to pay for it for a year which means I would only be able to monitor temperature and not have any extra features such as a

timer or pre-sets for certain foods. I was not intending on paying for this feature, so I had to find an alternative

Upgrade to a **Paid plan** to unlock **Triggers** feature

Monitor specific variables within your Things and receive notifications



By setting up Triggers, you can define conditions for when a variable's status change. Once the desired state is reached, the specified Actions will be executed, such as an email or a push notification sent to your defined pool of users, ensuring that everyone is promptly informed.

UPGRADE

LEARN MORE [↗](#)

11th December

I looked for alternatives and found Blynk cloud to be the platform I choose to develop my device on as you do not need to pay to implement events with the device. I started off setting up the Blynk by making data streams for the temperature for both Celsius and Fahrenheit so this device can appeal to anyone in the world regardless of which degrees they use. These data streams are essentially the same as the cloud variables I previously made on Arduino however they also come with virtual pins which will come in handy later for developing additional features that will be able to interact with the widgets. For both temperatures I set the minimum and maximum values of what the DS18B20 sensor supports and converted the Celsius to Fahrenheit when adding those. It is convenient that Blynk has built in units too so instantly it can identify the inputs from these data streams down the line in degrees.

Id	Name	Alias	Color	Pin	Data Type	Units	Is Raw	Min
1	TemperatureC	TemperatureC		V0	Integer	°C	false	-55
2	Temperature Fahrenheit	Temperature Fahrenheit		V1	Integer	°F	false	-67

The next step is to connect the previous code I had working to Blynk, unlike Arduino using a cloud agent to connect the board via WIFI with Blynk you connect to the cloud at runtime of the program.

FIRMWARE CONFIGURATION

```
#define BLYNK_TEMPLATE_ID "TMPL5HLCFeps9"
#define BLYNK_TEMPLATE_NAME "Smart Thermometer"
#define BLYNK_AUTH_TOKEN "vTMEZaOvMNT0UeaAzmlCh9xZPpVlVQng"
```

Template ID, Template Name, and AuthToken should be declared at the very top of the firmware code.

```
char auth[] = BLYNK_AUTH_TOKEN;

char ssid[] = "SHELL-483758_EXT";
char pass[] = "hkfKvfm3MQhk";
```

By pasting the firmware configuration at the top of the code and using the previous WIFI details I had alongside the authentication variable which uses the Authentication token I will be able to make a connection to Blynk.

```
23 void sendSensor()
24 {
25     sensors.requestTemperatures();
26
27     Serial.print("Celsius temperature: ");
28     Serial.print(sensors.getTempCByIndex(0));
29     Serial.print(" - Fahrenheit temperature: ");
30     Serial.println(sensors.getTempFByIndex(0));
31     int tempC=sensors.getTempCByIndex(0);
32     int tempF=sensors.getTempFByIndex(0);
33     delay(1000);
34
35     Blynk.virtualWrite(V0, tempC);
36     Blynk.virtualWrite(V1, tempF);
37     delay(500);
38 }
```

The code I previously had for displaying the temperature was more or less however I had to make use of the virtual pins in order to assign the data being read by the sensor so that it could be displayed on blynk. I had to look online for ways to make it work as getting the virtual pins to assign the readings to them was only the first step.

```
39 void setup()
40 {
41     Serial.begin(9600);
42     sensors.begin();
43
44     Blynk.begin(auth, ssid, pass);
45     timer.setInterval(100L, sendSensor);
46
47 }
48
49
50 void loop()
51 {
52     Blynk.run();
53     timer.run();
54 }
```

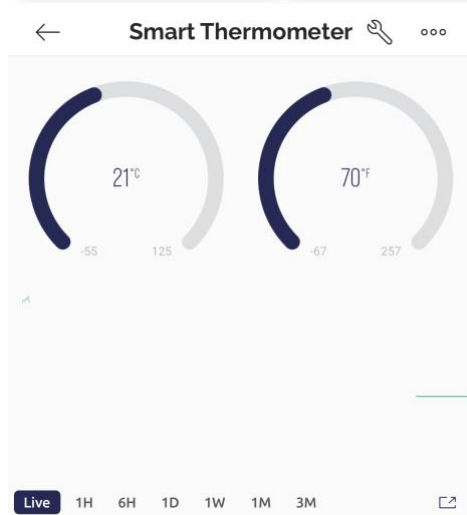
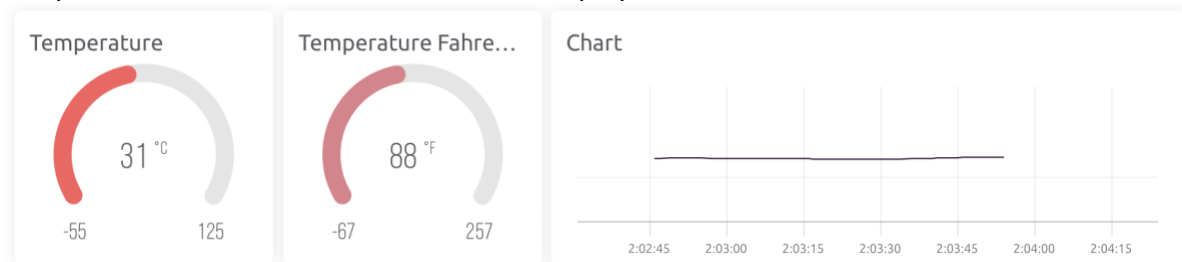
Using the blynk. begin method it allows the board to authenticate the connection by using the authentication key and the WIFI details. And in order to get consistent readings I set the method sensor to be called every 100 milliseconds in order to achieve real time temperature monitoring.

After running the following code, the board is now connecting to Blynk

 Smart Thermometer vTMEZaOvMNT0UeaAzmlCh9xZP... dominikjk@hotmail.com (you) Online

Now that a connection was made the next step is to create a dashboard to display the data. Just like Arduino blynk has its own widgets however they do have a paywall for the more

advanced ones, so I am limited to the ones that are provided for free. As I have two temperature types I added two Gauges for both degrees, by assigning the appropriate data stream for each gauge whilst connected to the board the gauges display the temperature and update in real time. The blynk Gauges also have a feature that allows you to assign a colour gradient which darkens and lightens as it goes up, as can be seen below as the gauges are working and display the data and there is a colour based on that temperature. Another aspect that could be seen as convenient to tracking the temperature is how it is progressing over time whilst cooking and this can be achieved by adding a chart and assigning the temperature data stream to it in order to display it.



Blynk also allows for making a mobile dashboard, so I made one using the same widgets and assigning the same variables so that the same data is also displayed on the phone. In the real world this device would usually be used via a phone as opposed to a PC, so it is important that the dashboard works for it as much as on the browser.

Now that the dashboard is working the next step is to add events for the functionality so that the user can be alerted once the water is boiling as this is a crucial feature for this device. In the events tab on Blynk I set up the event for when the water is boiling

EVENT NAME: EVENT CODE:

TYPE: ☒ Info ☐ Warning ☐ Critical ☐ Content

DESCRIPTION (OPTIONAL):

Limit: Every message will trigger the event

Event will be sent to user only once per

This is how I set up the event for the boiling water, the lowest rate per notification is one second and then one minute so one minute is the more appropriate time for letting the user constantly know that the water is boiling. For the notification tab I set it so that it sends notifications directly to the device owner of the app.

Now that the event was set up, I had to research how to call the event, using a simple If statement and by calling the temperature variable I can make it so the event is called once it reached above a certain degrees.

```
if (tempC > 20) {  
  Blynk.logEvent("boiling_water");  
}
```

As tempC is assigned to the virtual pin for temperature it is just the case of setting a condition for when tempC is > 20 it will trigger the event. I've set it to 20 only for

testing purposes as if it works for 20 same way it will work for 100 degrees. Now when running the code and holding the sensor in my hand I am expecting it to successfully trigger the notification.






Blynk Notification
Smart Thermometer:
Boiling Water
Water is Boiling

After running the code, the event was successfully called and the appropriate notification was loaded notifying me that the temperature was above 20 degrees. So now the device had real

time temperature monitoring working and notifications for once the temperature would reach above 100 degrees the user could be notified of this is which I consider a crucial feature of this device for both safety and efficiency purposes.

12th December

The next functionality I need to implement is the timer for the device, out of all the widgets the most convenient one to allow the user to set the desired length of time for the timer was the widget slider as it can be easily set by sliding towards the desired time. Without researching anything I know that in order to get a timer to work I need to make a new data stream for it I set the min value as one minute and the max as 30 as no one realistically is cooking via a pot for more than that so it should be enough.

Id	Name	Alias	Color	Pin	Data Type	Units	Is Raw	Min
1	TemperatureC	TemperatureC		V0	Integer	°C	false	-55
2	Temperature Fahrenheit	Temperature Fahrenheit		V1	Integer	°F	false	-67
3	Timer	Timer		V2	Integer	min	false	1

The next step is getting the code for this working, the timer's virtual pin is set as V2, and this is what I will be using to receive the data from the widget and to eventually run the timer based on this input. I researched the possibility of getting this to work so I'll primarily need to code this myself. I started off by making a variable for the data stream and making variables that will be used for the logic behind the timer. I also added a new method which when called would call the notification to alert the user that the timer is done.

```
30 void sendTimerDoneEvent() {  
31   Blynk.logEvent("timer_done"); // Log the 'timer_done' event  
32 }
```

```

16  #define TIMER_DATASTREAM_PIN V2 // Assign the DataStream virtual pin
17
18  int timerDuration = 0; // Variable to store the timer duration in seconds
19  int sliderValue = 0;   // Variable to track the slider value

```

The following code is for receiving data from the widget via its virtual pin, I used the Timer's data stream pin as a parameter. Then I added a variable for converting the input which would be in minutes into seconds by multiplying it by 60 and assigning it to the timerDuration variable. The sliderValue variable updates the slider value and then via the if statement once there is an input greater than zero the timer starts counting down

```

53  BLYNK_WRITE(TIMER_DATASTREAM_PIN) {
54      timerDuration = param.asInt() * 60; // Convert minutes to seconds
55      sliderValue = param.asInt(); // Update slider value
56      if (timerDuration > 0) {
57          // Start the timer
58          previousMillis = millis();
59      }
60  }

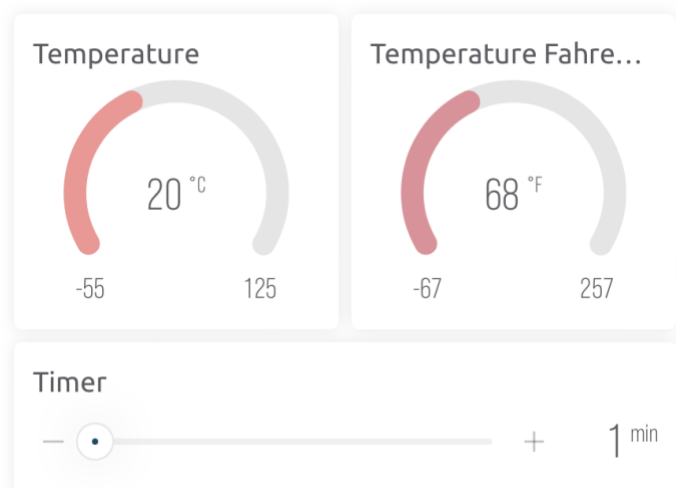
```

I then created the method for checking the timer, this code is supposed to check if the timer is greater than 0 and then decrement the timer every time it checks if the current millisecond counter is less than a second and then overwrite the virtual pin and the timer duration. Upon writing this code I am expecting the slider to successfully send the input and the timer will begin eventually sending the notification.

```

62  void checkTimer() {
63      if (timerDuration > 0) {
64          unsigned long currentMillis = millis();
65          if (currentMillis - previousMillis >= 1000) {
66              previousMillis = currentMillis;
67              timerDuration--; // Decrement the timer duration
68          }
69          Blynk.virtualWrite(TIMER_DATASTREAM_PIN, timerDuration);

```



After running the slider, it just stayed at one minute and didn't actually function I think it has to be a problem with my code despite having the actions for the timer to countdown it isn't being called correctly.

```

71     if (timerDuration == 0) {
72         sendTimerDoneEvent();
73     }
74 }

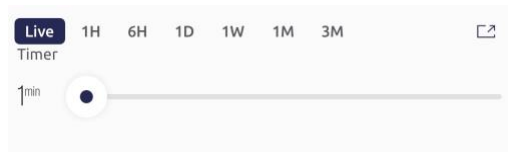
```

```

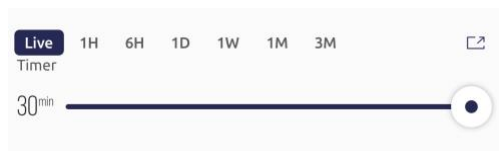
78 void setup() {
79     Serial.begin(9600);
80     sensors.begin();
81
82     Blynk.begin(auth, ssid, pass);
83     timer.setInterval(1000L, sendSensor);
84     timer.setInterval(1000L, checkTimer); // Check the timer every second
85 }

```

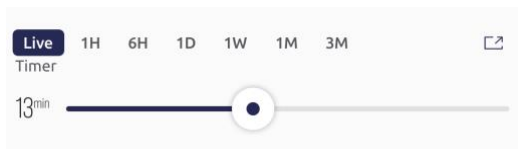
After adding the following code now, I am expecting the timer to work as the method for calling the notification is being called once the timer is 0. I tested this feature on the mobile browser



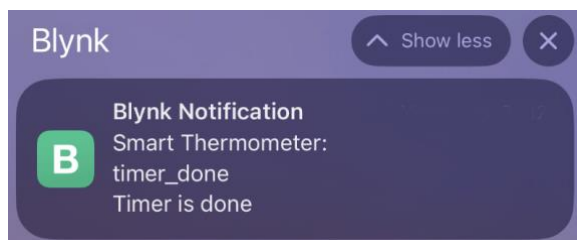
I set the timer to 1 minute to test it



After setting the timer went straight to 30 minutes and I thought it wasn't working



After 30 seconds the slider started counting down the seconds although it isn't what I'm going for the timer was functioning.



After one minute passed and the slider went all the way down to 1 minute the notification appeared meaning the timer is functioning.

15th December

Another feature I want to implement is having pre-sets for certain foods eg pasta and eggs would have a different timer pre-set. Despite having the timer feature which is very easy to use having a button the user could press to set the timer would make it even more convenient to use. Like the timer feature I think I could do this by setting up buttons with a data stream I believe even the timer data stream would work however I am not sure if Blynk will be able to support this by pushing the desired value every time the button is pressed.

Switch Settings ⓘ

TITLE (OPTIONAL)

Pasta Al Dente

Datastream

Timer (V2)

ON VALUE

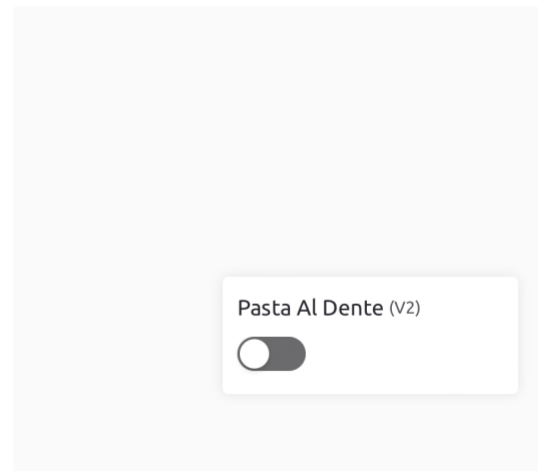
9

OFF VALUE

0

☐ Show on/off labels

☐ Hide widget name



Blynk doesn't have any button widgets like Arduino does however I instead added a slider and for the value assigned 9 minutes which is in the middle of what the recommended time is for cooking pasta al dente. The next step is to implement this in the code.

I created the following pseudocode for how I think it would work.

WHEN BLYNK SWITCH CHANGES (BLYNK_WRITE(TIMER_SWITCH_PIN)):

IF SWITCH STATE IS ON:

SET TIMER DURATION TO 9 MINUTES

UPDATE TIMER DATASTREAM WITH VALUE 9

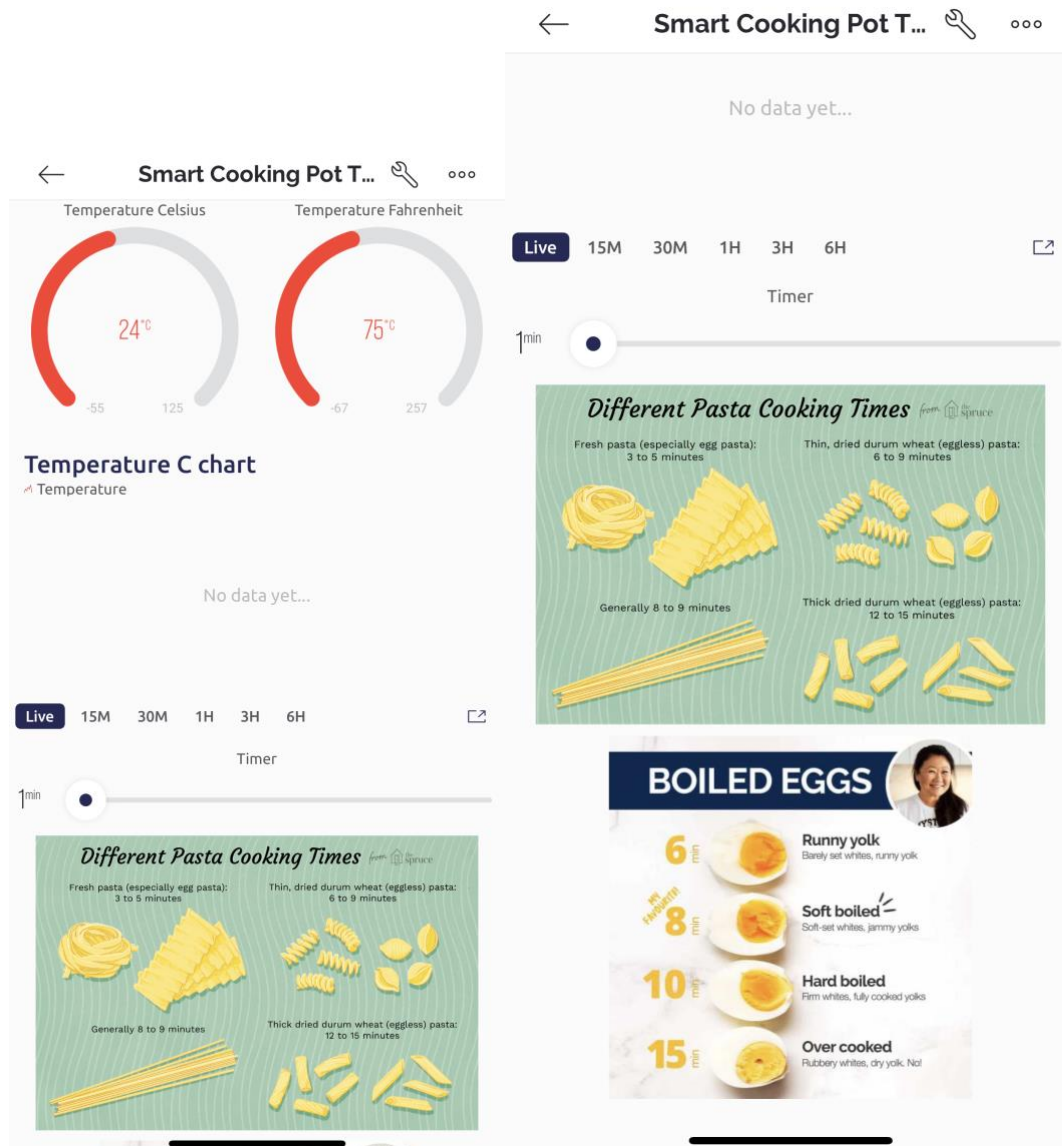
ELSE:

RESET TIMER DURATION TO 0

UPDATE TIMER DATASTREAM WITH VALUE 0

Unfortunately, I did not have enough time to implement the slider feature so the device will still be able to work for any food however the user will have to set the timer themselves.

I had my housemate Ollie test the dashboard and give feedback based on its current design and he said the Gauges could be bigger alongside the chart so I took this into account and made them larger I also previously didn't change the colour associated with the gauges and they were a solid black so he suggested changing it to Red could make it cleaner and Blynk has a feature where the gauge can use a gradient and the colour will be darker when the temperature is low and darker when it is hot. Taking also into consideration the fact I couldn't implement the pre-set feature in time using the widgets I ended up adding images which show the recommended cooking times for Pasta and Eggs this at least provided the user with info which they can then set the timer based on what is recommended for the optimum taste. The website dashboard is the same as the mobile dashboard however this device would be primarily used on the app as opposed to the browser however the timer functionality worked for both as the same data stream was used. Below is the final mobile dashboard:



Reflection:

How I approached this piece of work was by first planning out the necessary sensors I would need to make the final device. After finding the appropriate sensor being the DS18B20 I had to first figure out how to connect it to the board so that it can read the temperature from it, I achieved this by finding the libraries for the sensors and figuring out how the code works. The next step was the first prototype for connecting the sensor to the internet and sending the data and accessing it via the web and not the serial monitor. After achieving this and realising this would not be a viable option, I had to find a solution which would ultimately make this device truly smart and Blynk was the right option for me. The next step was to make that connection to blynk which I successfully did by modifying the code and setting up all the appropriate events and data streams. After getting the data streams to work the next step was to create the dashboard so that the temperature can be monitored in real time. For each widget I assigned the correct data stream and needed to modify the code to act accordingly with the widgets. Once this was completed, I implemented a feature I considered crucial which was the timer. This took a lot of trial and error and although functional it doesn't display how I'd like it to via the slider however at least in the last 30

seconds the slider goes down indicating it is nearly done. Unfortunately, I couldn't implement the pre-set features I wanted for pasta (al dente) and egg's however the user will have to set the timer themselves. The parts I found easy were setting up the blynk dashboard and connecting it due to the 3 lines of code you need to paste at the top. Thanks to the library that comes with the DS18B20 sensor I could easily figure out the code for making that initial connection with all the example code it has. The harder parts were using HTML alongside Arduino as I had trouble figuring out a local address was needed; the hardest part was implementing the slider as there wasn't much help online, so I used a lot of trial and error to get it working. Unfortunately, in the end the pre-set feature was not developed however the dashboard provides information for the user to set the timer based on what type of pasta they have or how they want their eggs. If I were to do the work again, I would spend more time playing around with the widgets and trying to develop more features, I would also try to make it more for general cooking although this would be limited as an oven at 200 degrees would not be able to be used with the device due to the DS18B20 limitation. What I learned is how a better understanding how smart devices communicate and the different processes they carry out to function. I also learned how Arduino boards work and now have some Arduino coding practice and know more about how sensors connect to the board and how that data is then eventually sent after using a cloud service. I would like specific feedback on the features that I have implemented, both allow the device to be completely functional despite the Timer feature not displaying the correct amount of time left it still functions and I am quite happy after spending a while figuring out how to get it to work. I feel quite proud of the work I managed to get a device which fulfils its purpose and can actually be used in the real world.

The following is the full code for my device.

```
#define BLYNK_TEMPLATE_ID "TMPL5SFQIga0Y"
#define BLYNK_TEMPLATE_NAME "Smart Cooking Pot Thermometer"
#define BLYNK_AUTH_TOKEN "GrWIBjYx_UQ25mhJJ4Gmy7epgT5bpdOJ"

#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include <OneWire.h>
#include <DallasTemperature.h>

#define oneWireBus D8
OneWire oneWire(oneWireBus);
DallasTemperature sensors(&oneWire);

#define TIMER_DATASTREAM_PIN V2 // Assign DataStream virtual pin

int timerDuration = 0; // Variable to store the timer duration in seconds
int sliderValue = 0; // Variable to track the slider value
```

```
unsigned long previousMillis = 0; // Variable to store the previous time
```

```
char auth[] = BLYNK_AUTH_TOKEN;
```

```
//char ssid[] = "SHELL-483758_EXT";
```

```
//char pass[] = "hkfKvfM3MQhk";
```

```
char ssid[] = "microlab_IoT";
```

```
char pass[] = "shibboleet";
```

```
BlynkTimer timer;
```

```
void sendTimerDoneEvent() {
```

```
    Blynk.logEvent("timer_done"); // Log the timer done event
```

```
}
```

```
void sendSensor()
```

```
{
```

```
    sensors.requestTemperatures();
```

```
    Serial.print("Celsius temperature: ");
```

```
    Serial.print(sensors.getTempCByIndex(0));
```

```
    Serial.print(" - Fahrenheit temperature: ");
```

```
    Serial.println(sensors.getTempFByIndex(0));
```

```
    int tempC = sensors.getTempCByIndex(0);
```

```
    int tempF = sensors.getTempFByIndex(0);
```

```
    if (tempC > 100) {
```

```
        Blynk.logEvent("boiling_water");
```

```
    }
```

```
    Blynk.virtualWrite(V0, tempC);
```

```
    Blynk.virtualWrite(V1, tempF);
```

```
}
```

```
BLYNK_WRITE(TIMER_DATASTREAM_PIN) {
```

```
    timerDuration = param.asInt() * 60; // Convert minutes to seconds
```

```
    sliderValue = param.asInt(); // Update slider value
```

```
    if (timerDuration > 0) {
```

```
        // Start the timer
```



```
    previousMillis = millis();
  }
}

void checkTimer() {
  if (timerDuration > 0) {
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= 1000) {
      previousMillis = currentMillis;
      timerDuration--; // Decrement the timer duration

      Blynk.virtualWrite(TIMER_DATASTREAM_PIN, timerDuration);

      if (timerDuration == 0) {
        sendTimerDoneEvent();
      }
    }
  }
}

void setup() {
  Serial.begin(9600);
  sensors.begin();

  Blynk.begin(auth, ssid, pass);
  timer.setInterval(1000L, sendSensor);
  timer.setInterval(1000L, checkTimer); // Check the timer every second
}

void loop() {
  Blynk.run();
  timer.run();
}
```