Literature on

“IoT project 1: ESP32 based Weight scale”

Hof University of Applied Sciences

Master’s in Software Engineering for Industrial Application (M. Eng.)

Hochschule Hof

University of Applied Sciences

Written by
Aatif Shaikh,
Landwehrstaße 03,
95028, HOF
Germany

ESP32 with HX711

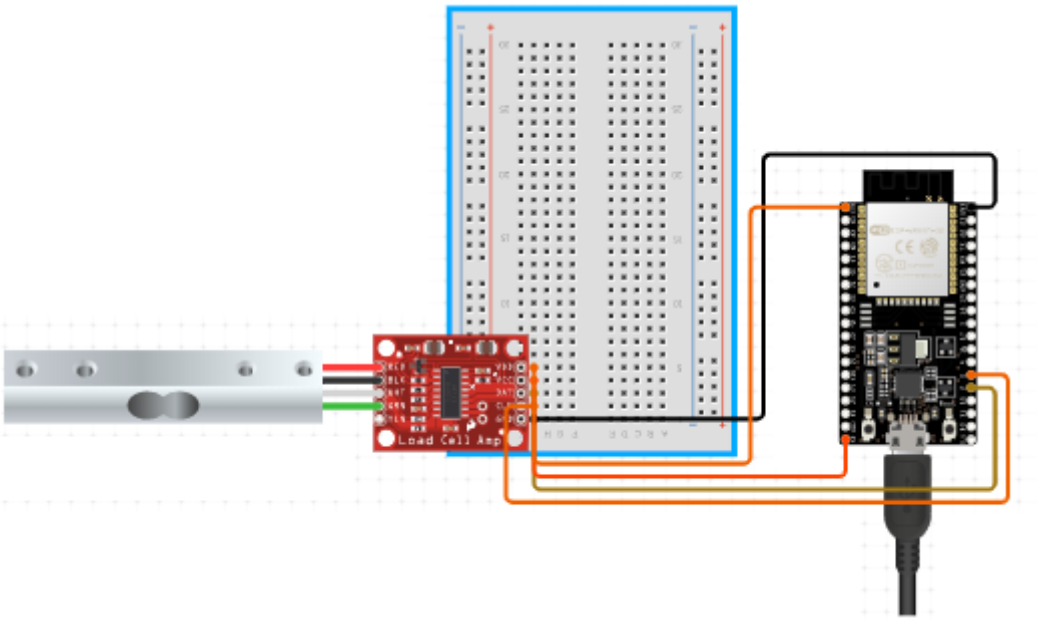


Image source: https://www.circuito.io/

In the Embedded (Electronics) world, there are many different types of sensors, modules, and devices available for the Arduino/ ESP family. They are simple and easy to use, and with the help of those, you can build pretty much any project. In this tutorial, we shall be building a very highly precise weight scale using HX711 and ESP32.

What is HX711 is?

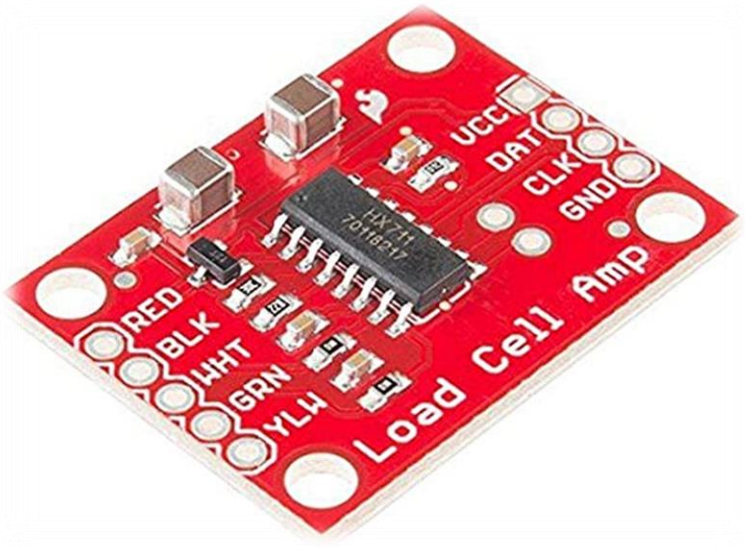


Image source: https://www.amazon.de/SparkFun-SEN-13879-W%C3%A4gezelle-Verst%C3%A4rker-HX711/dp/B00NPZ4CPG

The HX711 is a precision 24-bit ADC (analogue to digital converter). HX711 is specially designed for amplifying the small signals coming from the load cells (metallic bar). After that, it converts them into digital form and using any microcontroller or processor, final conversions (digital values) can be accessed.[1][2]

The resistive load cell is used to measure either a force or weight is applied to the bar. The HX711 converts the "Applied Force" into measurable electrical output. [1]

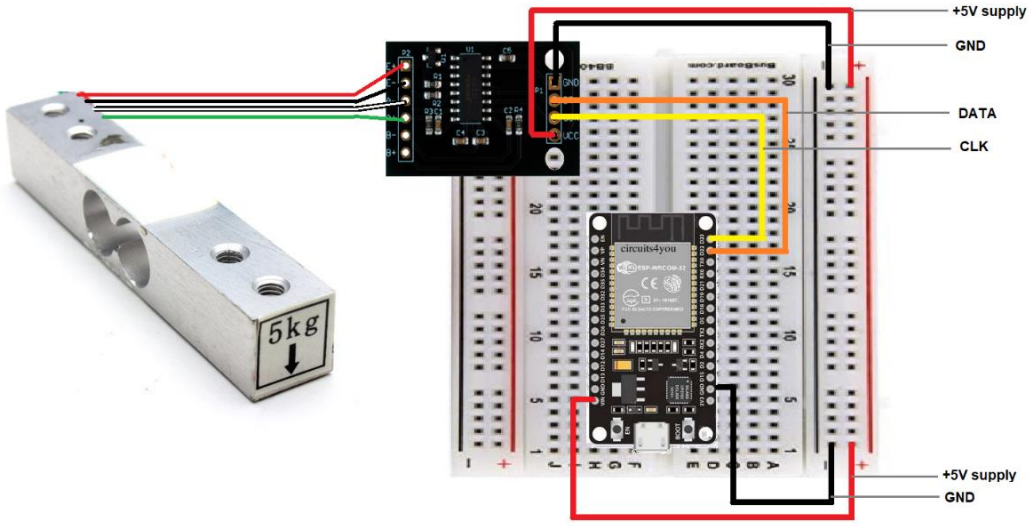
In a short definition, the HX711 is an electronic scale module whose basic working principle is to convert the measured changes in resistance value.[2]

The HX711 chip integrates a regulated power supply, an on-chip clock oscillator, and other peripheral circuits, which have the advantages of high integration, fast response, and strong anti-interference.[1]

The HX711 works on an I2C protocol, so, therefore, there is a 4-pin connector for CLK, DAT and supply. On the other side, you have to follow the colour code, so it's a bit easy to understand. The colour coding is Red (E+), Black (E-), White (A-), Green (A+) and yellow (B-). Sometimes, you will not find the yellow (B-) wire because it's not needed.[1]

Note* Based on your design (scaling machine), HX711 requires calibration. Therefore, you first need to download the calibration code. Once you have done with the calibration, you can download the main code with the calibration values added to it.

Connection Diagram



In the above diagram, you can see the overall connection diagram of the project. The VIN pin (on the ESP32) also supplies the +5V. So, you can connect the VIN pin of ESP32 to the supply line on the breadboard (Red-line on the corners). The ground (GND) pins available on the ESP32 are connected (common). Therefore, you can use any ground pin you want. GND next to 3.3V pin and VIN both is the same (Connect the GND to Black-line). Finally, you need to connect the Data (DAT) and Clock (CLK) pins, so you need to connect the Data (DAT) pin of HX711 to the D22 pin of ESP32 and the Clock (CLK) pin of HX711 to the D23 pin of ESP32.

In short and easy term:

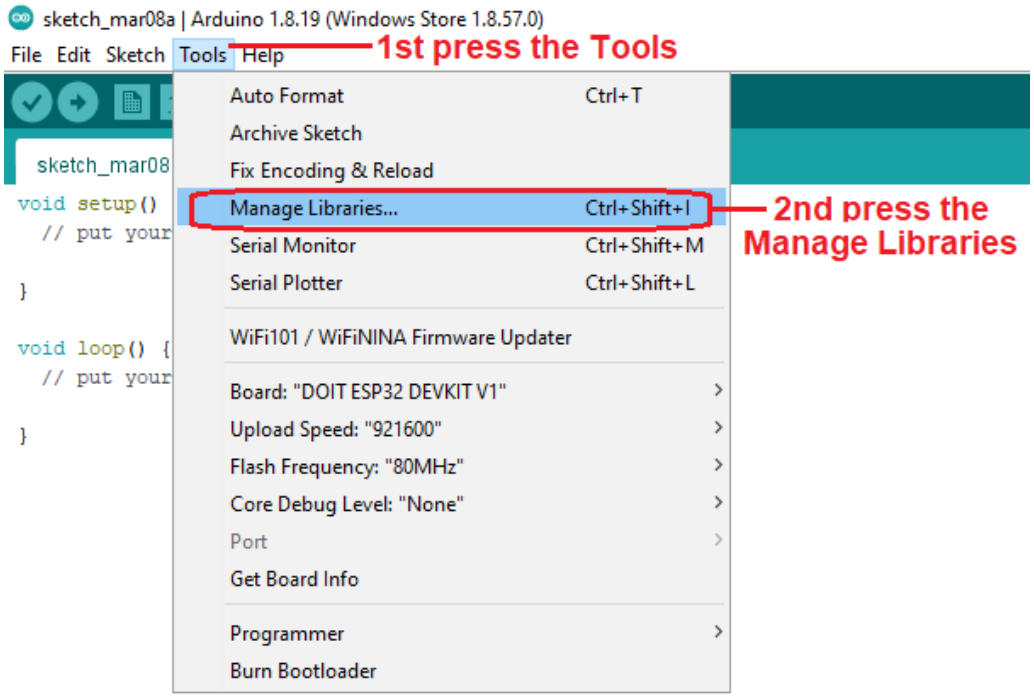
HX711 [GND] ←------→ ESP32 [GND]

HX711 [DT] ←------→ ESP32 [D22]

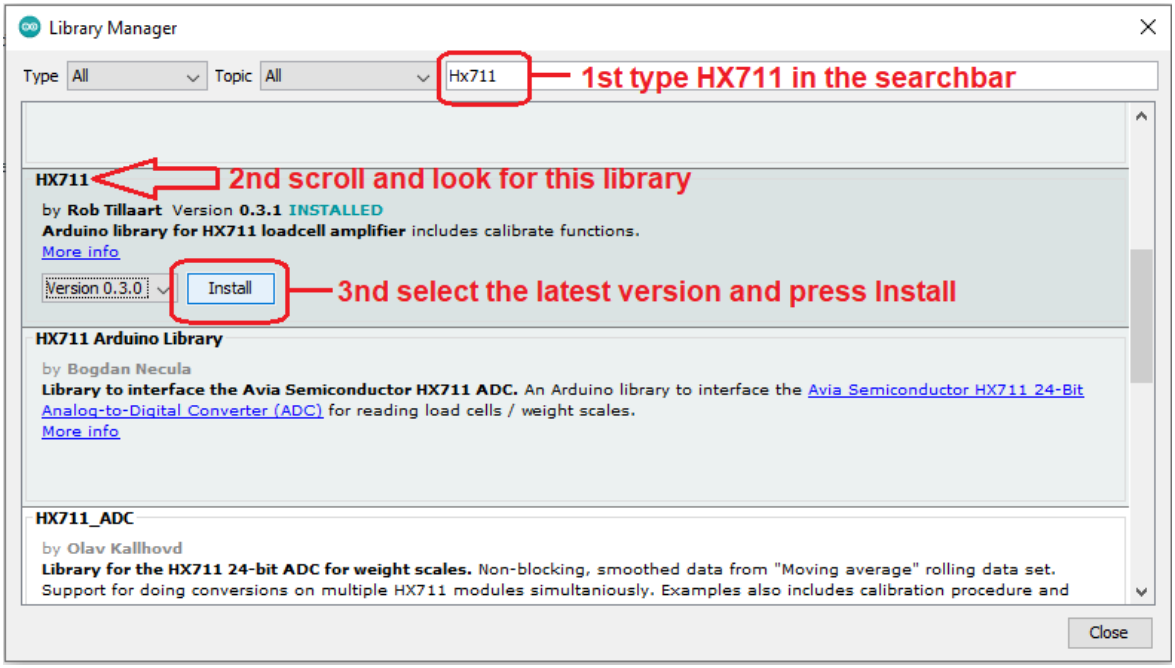
HX711 [CLK] ←------→ ESP32 [D23]

HX711 [VCC] ←------→ ESP32 [VIN]

Installing libraries for HX711



To add the libraries of HX711, you need to go to **Tools → Manage Libraries..**



Once the library manager window pop-up, enter **HX711** in the search bar. After that, scroll through the libraries and look for **HX711 by Rob Tillaart**. Select the latest version and press the Install.

Beginning of the code

As the Arduino IDE uses the C++ language. The first thing we need to do is add the necessary libraries and define the required definition. After that, initializing the peripherals and drivers are necessary. For this project, we will add HX711 and a WIFI library.

```
/**************************************************************************************
*  Libraries
**************************************************************************************/
 #include "HX711.h"
 #include <WiFi.h>         //wifi standard library
 #include <WiFiClient.h>   //wifi client library
 #include <WiFiAP.h>       //wifi host/ AP library

/**************************************************************************************
*  Defines
**************************************************************************************/
 #define DATA_PIN  22    //D22 pin on the ESP32
 #define CLK_PIN   23    //D23 pin on the ESP32
```

After that, we need to define the global variables.

```
/**************************************************************************************
 *  Global variables
 **************************************************************************************/
  /*set the wifi SSID and Password*/
  const char* wifiSSID     = "ESP32_WEIGHT";
  const char* wifiPassword = "SETyourOWNpassword";

  IPAddress  LocalIPAddress      (192,168,1,1); /*Set your Static IP address*/
  IPAddress  LocalGatewayAddress (192,168,1,1); /*Set your Gateway IP address*/
  IPAddress  LocalSubnetMask     (255,255,0,0); /*Set the subnet mask*/
  WiFiServer TCPServer(2222);    /*set the port number*/
  WiFiClient TCPClient = NULL ;  /*WIFI Server */
  int  TCPCleintStatus = 0;      /*To store the connection status of TCP*/
  int  TCPMessageCNT   = 0;      /*To store how many messages we have received*/
  char TCPTransmitBuffer[100];   /*Transmit buffer*/
  char TCPReceiveBuffer [100];   /*Receive buffer*/

  HX711 HX711vairable;         /*H711 object*/
  float CalibrationFactor = -187000; /*calibaeration factor*/
  float WeightInLbs  = 0 ;   /*variable to hold the weight in Lbs*/
  float WeightInKG   = 0 ;   /*variable to hold the weight in Kg*/
```

In the above code snip-it, you can see the defined variable. Each one holds a different value and parameter. The variable **wifiSSID** and **wifiPassword** holds the SSID of wifi and password. So, it's to change this SSID and Password by something different, so it doesn't intersect with other WIFI networks. To change the SSID and Password, you need to replace the string in between the double quotes ("ESP32_WEIGHT" -> "MyNewWiFi").

Variable **LocalIPAddress** and **TCPServer** hold the server IP address and port number. So after downloading the code, you can connect to this IP and port number on any TCP-IP application app.

Variable **CalibrationFactor** holds the calibration factor. Initially, you need to calibrate the device, and after calibrating the device, you'll get the updated values that you need to replace with the current value.

Finally, **WeightInLbs** and **WeightInKG** hold the weight value in Lbs and KG.

Setup of the code

In the below code snip-it, you can see the setup function, which is responsible for setting up all the peripherals. At the beginning of the code, the initialization of a serial terminal (**Serial.begin**) is necessary for debugging and managing the code during run-time. In the top right corner of Arduino IDE, you'll see an icon (magnifying glass). After downloading the code, you can press the magnifying glass icon to check print messages.

```
void setup( )
{
    Serial.begin(115200); /*initialize the serial port for printing the mesages*/

  if (! WiFi.softAPConfig(LocalIPAddress, LocalGatewayAddress, LocalSubnetMask ) )
     {  Serial.print("WIFI Static Configuration Failed!\n\r"); } /*Configures static address for wifi*/
 else
     { Serial.print("WIFI Static Configuration Success!\n\r"); }
  WiFi.softAP(wifiSSID, wifiPassword); /*start the wifi host*/
  IPAddress myIP = WiFi.softAPIP(   ); /*get our own ip address to confirm*/
  Serial.print("Server IP address: "); Serial.println(myIP); /*print to check if the IP address is properly set*/

  TCPServer.begin(  );   /*start the TCP server*/
  TCPClient.stop(   );   /*Close all the previous connections*/
  TCPCleintStatus = 0 ;  /*reset the tcp connection status*/
  TCPMessageCNT   = 0 ;  /*reset all receive and transmit buffer*/
  memset(TCPTransmitBuffer,0,sizeof(TCPTransmitBuffer)); /*reset the transmit buffer*/
  memset(TCPReceiveBuffer ,0,sizeof(TCPReceiveBuffer )); /*reset the receive buffer*/
  Serial.print("TCP Server Start!\n\r");

  Serial.print("Weight Sensor HX711 Init\n\r");
  HX711vairable.begin(DATA_PIN,CLK_PIN);      /*Initialize the HX711 port*/
  HX711vairable.set_scale(CalibrationFactor); /*set the caliberation factor*/
  HX711vairable.tare( );                      /*Reset the scale to zero*/
  long CheckCalibrationFactor = HX711vairable.read_average(); /*get the calibration factor*/
  Serial.print("Calibration factor: "); Serial.println(CheckCalibrationFactor); /*print the factor to confirm*/
}
```

After that, we will configure and start a WIFI network in host mode. Consequently, once you download and restart the ESP32, you'll be able to see the defined WIFI network on your phone and laptop.

Lastly, we will initialize the HX711 drivers and configure the initial tare and set the calibration value. As mentioned earlier, the HX711 require calibration. After downloading the code, we shall calibrate the scale first. In the code, we added some WIFI based commands through which we can adjust the calibration factor.

Infinite loop of the code

```
void loop( )
{

  WeightInLbs = HX711vairable.get_units() ;             /*read the weight in lbs*/
  WeightInKG  = HX711vairable.get_units() * 0.453592 ;  /*read the weight in kg*/
```

In the above code snip-it, you can see that after the "**setup( )"** function, "**loop( )"** function is defined. The "**loop( )"** function executes for an infinite amount of time. So, all commands or tasks inside the **loop( )** function will be executed infinitely. Inside the function, you can see we are reading the weight value in Lbs and KG and storing them in the local floating variable.

```
    if( TCPCleintStatus == 0     /*If the connection status is 0 and */
    && !TCPClient.connected( ) ) /*when there is no client is connected*/
        TCPClient = TCPServer.available( ); /*accept the new connection*/

    if(TCPClient)/*a proper connection is present*/
      {
        if ( TCPCleintStatus != 1 ) /*check if the connection was already connected or it's a new*/
           { TCPCleintStatus  = 1 ; /*if it's a new connection, change the status*/
             TCPClient.write("You are connected to ESP32 Server now!\n");/*Send the message/ feedback to client/ connection*/
             Serial.print("A New client connected!\n\r");
           }
```

After that, we need to handle the TCP server connection. In the beginning, we will wait for any new connection to connect to our TCP server. When we receive a request for a new connection, we will change the status of our client connection flag.

```
    if( TCPCleintStatus == 0     /*If the connection status is 0 and */
    && !TCPClient.connected( ) ) /*when there is no client is connected*/
        TCPClient = TCPServer.available( ); /*accept the new connection*/

    if(TCPClient)/*a proper connection is present*/
      {
        if ( TCPCleintStatus != 1 ) /*check if the connection was already connected or it's a new*/
           { TCPCleintStatus  = 1 ; /*if it's a new connection, change the status*/
             TCPClient.write("You are connected to ESP32 Server now!\n");/*Send the message/ feedback to client/ connection*/
             Serial.print("A New client connected!\n\r");
           }
```

Now, when ESP32 and a client is connected, we will wait for the client to send a message. Usually, in TCP client-server communication, we receive messages character by character (one character at a time). Therefore, after receiving the individual characters, we need to store them in a buffer array to form a complete message and later processing.

```
            if ( TCPClient.available( )) /*connection is still connected*/
               {
                 char c = TCPClient.read( ); /*read the message send by the client*/
                 TCPReceiveBuffer[TCPMessageCNT++] = c ;/*store the message from the client*/
```

A specific character is used to know when the message got completed. This character is known as NEWLINE (\n). Therefore, whenever we receive a NEWLINE character, it indicates that the message is now completed.

After that, to process the received message, we will use the standard string function, known as strstr( ), to compare the received string. Furthermore, a standard input/ output function called sprintf( ) is used to form the reply message. Based on every command, a different task will perform.

```
if ( c == (char)'\n' )/*check if the received message is completed*/
   { /*check and compare the message, if the user wants to read the weight*/
     if( strstr(TCPReceiveBuffer,(char *)"read weight") != 0 )
       { /*form a reply message to send it to client*/
         sprintf(TCPTransmitBuffer,"Weight KG=%.3f Weight LBS= %.3f\n",WeightInKG,WeightInLbs);     }
     /*check and compare the message, if the user wants to tare the weight*/
     else if( strstr(TCPReceiveBuffer,(char *)"tare weight") != 0 )
       { HX711vairable.tare( ); /*Reset the scale to zero*/
         /*form a reply message to send it to client*/
         sprintf(TCPTransmitBuffer,"tare weight to zero\n");   }
     /*check and compare the message, if the user wants to increase the calibration value*/
     else if( strstr(TCPReceiveBuffer,(char *)"calibration inc") != 0 )
       { CalibrationFactor = CalibrationFactor + 100; /*increase the calibration factor by 100*/
         HX711vairable.set_scale(CalibrationFactor); /*set the caliberation factor*/
         /*form a reply message to send it to client*/
         sprintf(TCPTransmitBuffer,"caliberation value increased = %.0f\n",CalibrationFactor); }
     /*check and compare the message, if the user wants to decreased the calibration value*/
     else if( strstr(TCPReceiveBuffer,(char *)"calibration dec") != 0 )
       { CalibrationFactor = CalibrationFactor - 100; /*decreased the calibration factor by 100*/
         HX711vairable.set_scale(CalibrationFactor);  /*set the caliberation factor*/
         /*form a reply message to send it to client*/
         sprintf(TCPTransmitBuffer,"calibration value decreased = %.0f\n",CalibrationFactor);  }
      else
       { /*if the user sends an unknown command*/
         sprintf(TCPTransmitBuffer,"unknown command\n"); }
```

Finally, we will send the transmit message to the client for feedback and clear all the buffer for new messages to receive and send.

```
                    Serial.print(TCPTransmitBuffer);
                    TCPClient.write(TCPTransmitBuffer); /*send the reply back to the client*/
                    memset(TCPTransmitBuffer,0,sizeof(TCPTransmitBuffer)); /*clear the transmit buffer*/
                    memset(TCPReceiveBuffer ,0, sizeof(TCPReceiveBuffer)); /*clear the receive buffer*/
                    TCPMessageCNT = 0 ;  /*reset the receive message count*/
                  }
```

At the end of the code, we shall be checking the client status continuously to update the current status of it. And if the client gets disconnected, we will stop the connection with the client so that we can accept a new client.

```
    if ( TCPClient.connected( )) /*continiously check and update the status of tcp connection*/
       { TCPCleintStatus = 1;  } /*if it's a new connection, change the status to set*/
    else
       { TCPCleintStatus = 0;   /*if it's not connected, change the status to reset*/
         TCPClient.stop( );    } /*stop the client connecction*/
```

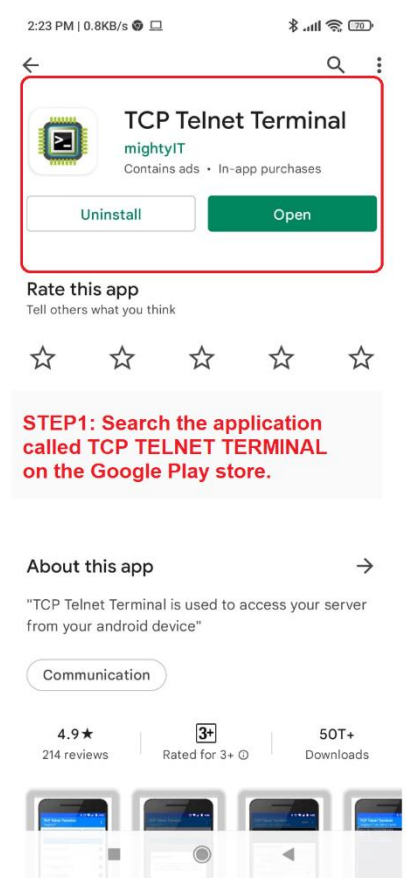
Complete code

You can also copy the code (below) and paste it in Arduino ID sketch, save it with whatever file name you like.

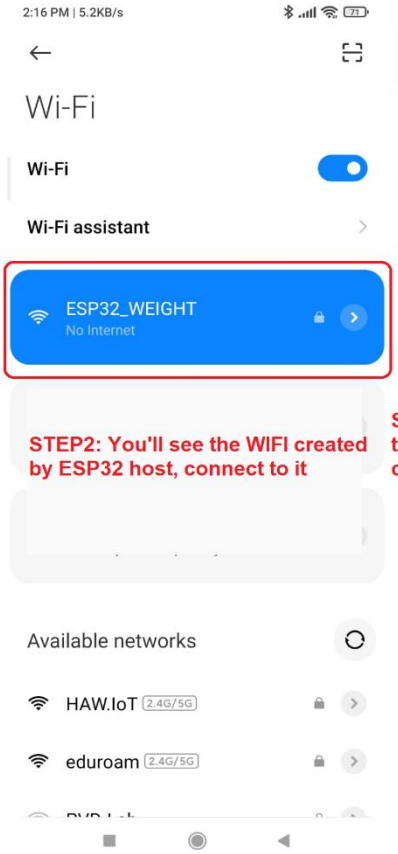
```
/*****************************************************************************
 *  Libraries
 *****************************************************************************/
 #include "HX711.h"
 #include <WiFi.h>        //wifi standard library
 #include <WiFiClient.h>  //wifi client library
 #include <WiFiAP.h>      //wifi host/ AP library

/*****************************************************************************
 *  Defines
 *****************************************************************************/
 #define DATA_PIN  22 //D22 pin on the ESP32
 #define CLK_PIN   23  //D23 pin on the ESP32

/*****************************************************************************
 *  Global variables
 *****************************************************************************/
 /*set the wifi SSID and Password*/
 const char* wifiSSID        = "ESP32_WEIGHT";
 const char* wifiPassword = "SETyourOWNpassword";

 IPAddress  LocalIPAddress      (192,168,1,1); /*Set your Static IP address*/
 IPAddress  LocalGatewayAddress (192,168,1,1); /*Set your Gateway IP address*/
 IPAddress  LocalSubnetMask     (255,255,0,0);  /*Set the subnet mask*/
 WiFiServer TCPServer(2222); /*set the port number*/
 WiFiClient TCPClient = NULL ;  /*WIFI Server */
 int  TCPCleintStatus = 0;     /*To store the connection status of TCP*/
 int  TCPMessageCNT   = 0;   /*To store how many messages we have received*/
 char TCPTransmitBuffer[100]; /*Transmit buffer*/
 char TCPReceiveBuffer [100]; /*Receive buffer*/

 HX711 HX711vairable;   /*H711 object*/
float CalibrationFactor = -187000; /*calibaeration factor*/
 float WeightInLbs  = 0 ;  /*variable to hold the weight in Lbs*/
 float WeightInKG   = 0 ; /*variable to hold the weight in Kg*/

/*****************************************************************
 * Name of the Function: setup
 * Parameter: None / Return: void
 *****************************************************************/
void setup( )
{
 /*initialize the serial port for printing the messages*/
 Serial.begin(115200);

 /*Configures static address for wifi*/
 if(! WiFi.softAPConfig(
      LocalIPAddress,
      LocalGatewayAddress,
      LocalSubnetMask ) )
     { Serial.print("WIFI Static Configuration Failed!\n\r"); }
 else
     {Serial.print("WIFI Static Configuration Success!\n\r"); }
  /*start the wifi host*/
  WiFi.softAP(wifiSSID, wifiPassword);
  /*get our own ip address to confirm*/
  IPAddress myIP = WiFi.softAPIP(   );
  /*print to check if the IP address is properly set*/
  Serial.print("Server IP address: ");
  Serial.println(myIP);

  TCPServer.begin(  );   /*start the TCP server*/
  TCPClient.stop(   );   /*Close all the previous connections*/
  TCPCleintStatus = 0 ;  /*reset the tcp connection status*/
  TCPMessageCNT   = 0 ;  /*reset all receive and transmit buffer*/
  memset(TCPTransmitBuffer,0,sizeof(TCPTransmitBuffer)); /*reset the transmit buffer*/
  memset(TCPReceiveBuffer ,0,sizeof(TCPReceiveBuffer )); /*reset the receive buffer*/
  Serial.print("TCP Server Start!\n\r");
```

```
  Serial.print("Weight Sensor HX711 Init\n\r");
  HX711vairable.begin(DATA_PIN,CLK_PIN);      /*Initialize the HX711 port*/
  HX711vairable.set_scale(CalibrationFactor); /*set the caliberation factor*/
  HX711vairable.tare( );                      /*Reset the scale to zero*/
  long CheckCalibrationFactor = HX711vairable.read_average(); /*get the calibration factor*/
  Serial.print("Calibration factor: "); Serial.println(CheckCalibrationFactor); /*print the factor to confirm*/
}

/*********************************************************************
 * Name of the Function: loop
 * Parameter: None / Return: void
 *********************************************************************/
void loop( )
{
 /*read the weight in lbs*/
 WeightInLbs = HX711vairable.get_units() ;
 /*read the weight in kg*/
 WeightInKG  = HX711vairable.get_units() * 0.453592 ;

   if( TCPCleintStatus == 0     /*If the connection status is 0 and */
   && !TCPClient.connected( ) ) /*when there is no client is connected*/
      TCPClient = TCPServer.available( ); /*accept the new connection*/

   if(TCPClient)/*a proper connection is present*/
     {
      if ( TCPCleintStatus != 1 ) /*check if the connection was already connected or it's a new*/
        { TCPCleintStatus  = 1 ; /*if it's a new connection, change the status*/
          TCPClient.write("You are connected to ESP32 Server now!\n");/*Send the message/ feedback to client/ connection*/
          Serial.print("A New client connected!\n\r");
        }

       /*connection is still connected*/
       if ( TCPClient.available( ))
         {
          char c = TCPClient.read( ); /*read the message send by the client*/
          TCPReceiveBuffer[TCPMessageCNT++] = c ;/*store the message from the client*/

          if ( c == (char)'\n' )/*check if the received message is completed*/
            { /*check and compare the message, if the user wants to read the weight*/
              if( strstr(TCPReceiveBuffer,(char *)"read weight") != 0 )
                { /*form a reply message to send it to client*/
                  sprintf(TCPTransmitBuffer,"Weight KG=%.3f Weight LBS= %.3f\n",WeightInKG,WeightInLbs);     }
              /*check and compare the message, if the user wants to tare the weight*/
              else if( strstr(TCPReceiveBuffer,(char *)"tare weight") != 0 )
                { HX711vairable.tare( ); /*Reset the scale to zero*/
                  /*form a reply message to send it to client*/
                  sprintf(TCPTransmitBuffer,"tare weight to zero\n");   }
              /*check and compare the message, if the user wants to increase the calibration value*/
              else if( strstr(TCPReceiveBuffer,(char *)"calibration inc") != 0 )
                { CalibrationFactor = CalibrationFactor + 100; /*increase the calibration factor by 100*/
                  HX711vairable.set_scale(CalibrationFactor); /*set the caliberation factor*/
                  /*form a reply message to send it to client*/
                  sprintf(TCPTransmitBuffer,"caliberation value increased = %.0f\n",CalibrationFactor); }
              /*check and compare the message, if the user wants to decreased the calibration value*/
              else if( strstr(TCPReceiveBuffer,(char *)"calibration dec") != 0 )
                { CalibrationFactor = CalibrationFactor - 100; /*decreased the calibration factor by 100*/
                  HX711vairable.set_scale(CalibrationFactor);  /*set the caliberation factor*/
                  /*form a reply message to send it to client*/
                  sprintf(TCPTransmitBuffer,"calibration value decreased = %.0f\n",CalibrationFactor);  }
               else
                { /*if the user sends an unknown command*/
                  sprintf(TCPTransmitBuffer,"unknown command\n"); }

                Serial.print(TCPTransmitBuffer);
                TCPClient.write(TCPTransmitBuffer); /*send the reply back to the client*/
                memset(TCPTransmitBuffer,0,sizeof(TCPTransmitBuffer)); /*clear the transmit buffer*/
                memset(TCPReceiveBuffer ,0, sizeof(TCPReceiveBuffer)); /*clear the receive buffer*/
                TCPMessageCNT = 0 ;  /*reset the receive message count*/
            }
          }
       }

  if ( TCPClient.connected( )) /*continuously check and update the status of tcp connection*/
    { TCPCleintStatus = 1;  } /*if it's a new connection, change the status to set*/
  else
    { TCPCleintStatus = 0;    /*if it's not connected, change the status to reset*/
      TCPClient.stop( );    } /*stop the client connection*/

}
```

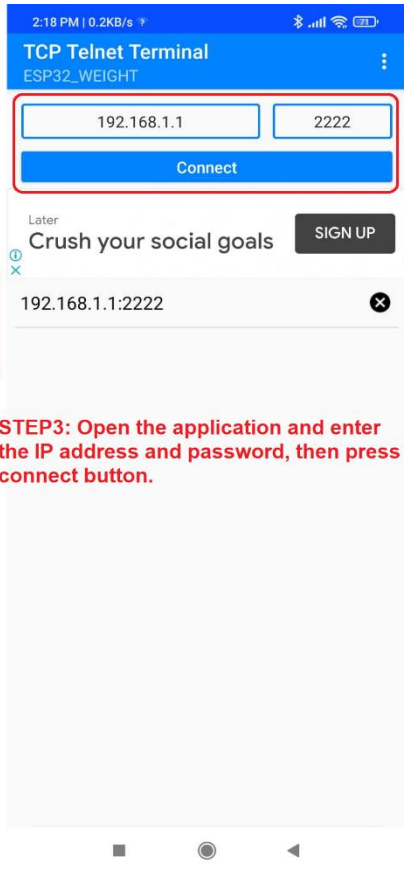
Reading Though Android application



STEP1: Search the application called TCP TELNET TERMINAL on the Google Play store.



STEP2: You'll see the WIFI created by ESP32 host, connect to it



STEP3: Open the application and enter the IP address and password, then press connect button.