Story

Summary

This is a WiFI-enabled weight scale, meant to primarily weight food and track calories. This is useful if you are on a diet, if you track your calorie intake for maintaining your weight or if you are into fitness and simply want to improve your results by keeping an eye on nutrition.

It can weight food with a very good precision up to a 10th of a gram, calculate energy in calories, display this info on the LCD, and using the RESTful API you can transfer the weight information, adjust settings or control the scale remotely.

Rationale

The reason I started this project was because I wanted to help my wife with her food tracking. We both have FitBit trackers and we like the way FitBit helps with these objectives. They are great at tracking energy expediture, but not so great at logging food. Because of some annoying bugs, lack of features and incomplete food database, I just created a web app from where we log our food, then this data is sent to FitBit via their REST API. Basically I created a better interface for it allowing us to create a shared food database. I am not currently using that web app to log my food (not on diet anymore), even if I still wear the FitBit tracker, but my wife likes to use it in a daily basis (even if she's only trying to maintain weight).

The way we did it was by weighting food with a conventional kitchen scale then just enter the weight manually into the web app. We had no info on how many calories that food actually had until we entered the info on the phone. So I told my wife: "What do you say if I tell you I can make you a scale that is able to log your food to FitBit, so you don't need to enter that weight manually, and and you will be able to see in real time the calories so you can add or remove stuff to keep thing under budget ? :). " Initially I wanted to also display the energy consumed/left on that day, but I decided to include minimal information on the scale LCD.

I had to decide between buying a load cell and create my own case or buy a cheap scale, and use it for the load cell inside and the stable base.

I chose to buy a cheap scale, from which I only kept the load cell and the case with push buttons. You can decide to create your own case if you think it looks or works better. I am pretty happy with the result, even if it doesn't look so nice, but hey, it's a DIY thing...

Product manual

Powering on/off

There is a mechanical on/off switch in the right side, next to the charge port. Make sure this is switched on.

You can power the scale on by short pressing the red (left) button.

To power off long press the black (right) button for at least 3 seconds.

You can also power off from the API:

`POST https://healthzuilla-scale/api/device/poweroff`

When powering on, the logo LED lights up bright, then after network connection it dims to about 20%, then completely dims off in 5 seconds.

Power saving: Auto-power off

The scale powers off automatically if inactive for 90 seconds. Inactivity means, no weight changes detected during that period. In addition, as a safeguard for the battery, the scale has a general power off timer set to 5 minutes, no matter if active or not. These timers are customizable through the API via /api/settings.

New in 2022.2: If you don’t build the fancy latching circuitry the scale will turn off most of it’s power sources and enter a deep sleep to conserve energy. With the latching circuit, everything will be off.

Charging

The scale contains a 1000 mAh rechargeable LiIon battery. When fully charged, the scale should typically operate for 1-2 months. A charge level indicator is present on the LCD. When the charge level gets under 15% , the internal LED will blink continuously and it is recommended to plug the charger in the right side USB socket. You can use any USB charger, but a 5V/1A charger is recommended.

This battery should be fully charged in 1h45m.

During charge, a red led will be lit. When fully charged a blue led lights.

There is a short circuit protection and over-charge protection when using an USB charger.

Maximum input voltage: 8V (the USB port is spec'ed for 5V with a 5% error margin).

Word of caution

Li-Ion batteries can be dangerous, especially old ones. If you see any sign of it getting hot (should not even get warm) or swelling, stop charging and replace it with a new one. You can replace it with 1000mAh - 2000mAh battery, never lower for the given charging circuit.

Do not try to charge the battery by plugging an USB cable in the esp8266 usb port, and do not connect the charger in the charge port while an USB cable is connected to esp8266 for debugging. That port is only for debugging and programming, and should only be used with the mechanical switch in the off position. *By not closing the switch when using this port, you will feed current to the battery, bypassing the charging circuit, which prevents over-charge and you have a high chance of damaging the battery.* ***Over-charging a Lithium-Ion battery can cause fire and explosion !***

> Even if a computer supplies low current on the USB ports (typically 250mA), so the current would be ok, the voltage would be too high for a Li-Ion battery (it will get somewhere between 4.8V - 5V since it passes through a Schottky diode) while the absolute maximum voltage for a Li-Ion should be 4.2V.

Networking

The scale can connect to an existing WiFi b/g/n network or it can act as a hotspot when in network setup mode.

Default network settings:

- IP: 192.168.1.11
- Gateway: 192.168.1.1
- Subnet mask: 255.255.255.0

When the scale starts, it tries to connect to the last known network. The LCD will display "Connecting to <SSID>" and a progress indicator is shown. If the connection is successful, you will see a signal strength indicator in the top right side of the LCD. On connect error, there will be an indicator with strikeout WiFi. You can still use the scale with no WiFi, but only in basic weighting mode.

Network Configuration Portal

For the first time, if the SSID of your WiFi or authentication changes you need to reconfigure the connection settings. You can start the configuration portal by double-pressing the reset button of the Wemos R1 Mini chip. With the scale turned on, you basically need to press the reset button twice in a 5 second interval.

When entering this mode, the scale will blink the logo LED twice, and display "*Configure me*" on the screen.

To use the configuration portal, connect to the public "Healthzuilla-Scale" WiFI network the scale creates. Then access the portal: http://192.168.4.1

If you don't configure the scale, there is a timeout of 5 minutes to use the portal.

From the config portal you can choose a SSID, set the password and the authentication type. Every time you change the settings, the DHCP client will be turned on. You need to use the API to disable DHCP by setting *useStaticIP* to *false* and a static IP configuration.

> The previous static IP settings are disabled and DHCP enabled to be able to connect to that network as it might conflict with the IP settings you already have. Once connected, it's recommended to set a static IP, so you know how to access the scale with the API. Or you can instruct the router to always assign the same IP for the scale's physical network address.

Using the scale

After powering on the scale, place the item on the green weighting area when the scale has finished initialization and display shows 0g. The weight is sampled at a 1 second interval and shown on the display.

To have the scale display both weight and calories, use the API to send the food info.

```
POST https://healthzuilla-scale/api/weight
{“foodId”: 123, “name”: “Banana”}
```

The scale will display the food name, weight and calories in real-time. The scale will remember that food until is powered off or you send another food info.

Tare function

The tare function resets the zero of the scale display for example when an empty container is placed on the weighing platform, in order to subsequently display only the weight of the contents of the container. This actually works by zeroing what's currently on the scale, so you can use this function when adding different foods in the same recipient, but want to weight them individually.

The scale always starts by doing tare, so if you place an empty plate before starting the scale, it will show 0.

Display back-light

The LCD back-light is controlled automatically depending on the ambient illumination. There is a sensor on the front of the scale, in the left side of the button button.

Technical information

This project requires good soldering skills, some background in creating simpler circuits, basic understanding of electronics and some experience with Arduino IDE. If you are new to this, I recommend trying a simpler project first, just to get the hang of it.

Power circuit

I recommend a mechanical switch on the positive wire of the battery, in case you want to cut the power from the battery. This circuit relies on a soft switch (a push button) so the scale is able to turn itself off. With only the mechanical switch this wouldn't be possible.

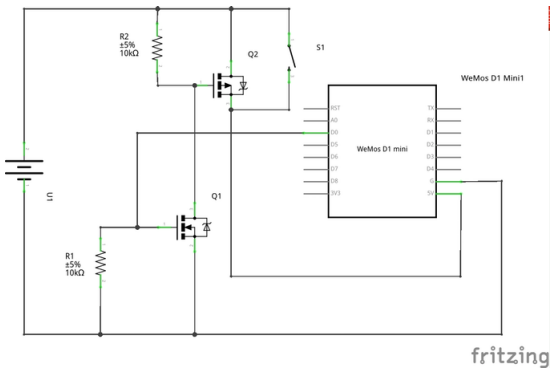
The current from the battery is filtered by a 47uF bulk electrolytic capacitor (C2) and a 100nF decoupling ceramic capacitor (C1)., both in parallel with the input from the battery The bulk capacitor is very important because it acts like a reservoir of energy for the hungry esp8266 chip and is essential at boot. esp8266 draws 70-150 mA in bursts, when it boots and can create spikes of 300-400mA. The voltage at the regulator input drops as a result. If the voltage drops too much, then it might cause a brownout. The bulk capacitor will smooth out these spikes and avoid voltage drops at the regulator input, offering a buffer of energy for the regulator. The smaller ceramic decoupling capacitor helps eliminate high frequency noise generated by the WiFi operation, and offers faster response in those cases.

At the regulator output, so at HX711 and LCD input, two decoupling ceramic capacitors are used, one of 100nF (C4) and the other of 10nF (C5). They help isolate these components, especially the load cell amplifier (HX711) which should get a stable voltage, and they suppress the high frequency noise trying to cover a wider range of frequencies and provide instant energy reserves when needed.

The soft latching circuit uses one P Channel Mosfet (NDB6020P) - Q2, one N Channel Mosfet (2N7000) - Q1, and two 10K resistors (R1 and R2).

> You can use any P Channel Logic Level Mosfet instead of NDB6020P able to provide at least 500mA at -3.3V,. This one is rated for up to 24 Amps, so it's a bit bulky, but didn't had a smaller one. You can also replace the N Channel

> Mosfet with a similar N Channel Mosfet or even a NPN transistor. If you replace it with a transistor, connect the base of the transistor through the 10K resistor to limit the base current. The advantage in using a Mosfet is much lower power consumption, and it only consumes current while the scale is on



Latching circuit schematic

Wemos D1 Mini is powered by the red (left) push button acting as a switch. As soon as it starts up, it puts digital pin GPIO16 to HIGH, feeding 3.3V into the N Channel Mosfet Gate, allowing the current to flow from drain to source on the negative side. The gate of Q1 is pulled down by R1 so the FET does not conduct initially. And that makes Q2 conduct between drain and source, as the Q2's gate is now pulled to ground. Q2 is a P Channel Mosfet and works the other way around - it is pulled high by the 10K resistor R2 by default and only allows current to flow when no voltage is applied to it's gate.

The micro-controller can kill it's own power by putting GPIO16 to LOW, removing voltage from Q1 gate, Q1 is now pulled LOW by R1, thus, interrupting the path to ground to Q2's gate, and Q2 is again pulled high by R2.

Physical Buttons

The scale has two physical push buttons. There is also a switch used for intrerrupting the power from battery to the connection board.

The first push button positioned in the left can’t be read using the micro-controller. It is only meant as a power-on switch. After the micro-controller is powered on, this button doesn’t have any purpose. I preffered a simple approach, where I turn on power from a button and I turn off power either from software or by reading another button. You could make the button on on snort press/off on long press, but a more complex circuit is needed.

The second button positioned in the right has a dual role:

- turn off the scale with a long press
- tare the weight with a short press

It was simply easier for me to separate the on and off buttons than trying to reuse the “on” button.

Basically only the second button is read with the micro-controller. To distinquish between short press and long presses I just store the time at which the button changed state and measure if enough time has elapsed to consider a long press (3sec). You can study the code in the function handleTareButton() which is executed in loop().

Scale calibration

It is very important to calibrate the scale before doing anything. For this a so called *calibration factor* is used. This calibration factor can be determined by the scale if you put the scale in *calibration mode*. To put the scale in calibration mode, use the HTTP API:

```
POST https://healthzuilla-scale/api/calibrate
```

The calibration mode will ask you to place a known weight on the scale. As soon as it detects the weight, it calculates the calibration factor, applies it, saves it into EEPROM for the next time it starts, and get's ready again to be used.

The known weight is set to 152g. This might sound strage, but it's actually the weight of my Samsung S7 phone. It's very convenient to use the phone itself to calibrate the scale. This value is taken from the specs (I guess they have measured that with a calibrated scale :)) and I confirmed the proper calibration by weighting several coins. Coins are also something you can use as a calibration weight, because they all have an official weight, but you need a bunch of them to weight at least 50g to make a kindof accurate calibration.

Reading weight

For measuring weight we need two components

1. **Load Cell**
   I am using a straight bar load cell also called a strain gauge
2. **Load Cell amplifier**
   I am using the a HX711 amplifier module

This straight bar load cell used is rated at 5kg, this means it can translate up to 5kg of pressure (force) into an electrical signal. The load cell is able to measure the electrical resistance that changes in response to, and proportional of, the strain (e.g. pressure or force).

More info on load cells.

To be able to read those small fluctuations in resistance, you need either a very precise and expensive instrumentation, or a good amplifier like HX711, which can convert those small changes into something readable.

The readings should be pretty accurate, as the load cell amplifier used (HX711) contains a 24 bit ADC. To get accurate results, make sure the scale has a stable base. You can use rubber legs.

The four wires coming out from the wheatstone bridge on the load cell are usually:

- •Excitation+ (E+) or VCC is red
- •Excitation- (E-) or ground is black.
- •Output+ (O+), Signal+ (S+)+ or Amplifier+ (A+) is white
- •O-, S-, or A- is green or blue

The white and blue wires coming out of the cheap scale used were actually reversed, but I noticed that because of the labels they had on the existing PCB.

I have read that HX711 works better with 5V, but I got satisfactory results with 3.3V. Your module might have dedicated power inputs for analog and digital supply, such as the one from Sparkfun.This separation allows to power the analog part with 5V and the logic part with 3.3V so you can use a 3.3V micro-controller. My module didn't had those options.

Sparkfun tutorial on using HX711.

The pins labeled **DAT** or **DOUT** and **CLK** can be connected to any digital pin. I used **GPIO5** as **DOUT** and **GPIO4** as **CLK.**

The library used should talke care of the communication. I used *bogde/HX711* as library., which most people are using
After several trials, the best algorithm of reading weight I could came is:

- read scale at every second in a timer
- at each reading get 7 samples and use the average
- use a Moving Average algorithm when the weight stabilizes to reduce abnormal variations, and improve accuracy. The moving average is based on the history of those 1 sec readings.

You can study the code inside the function *readScale()*.

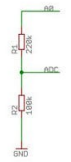
You can retrieve the weight using the API:

Reading battery state of charge

The state of charge of the Li-Ion battery is calculated based on the voltage. I assembled a table of charge percentage for different voltage values. You will find different tables online, more or less accurate, what I did was to look at the SoC vs voltage curve from a Li-Ion battery datasheet and try to follow it. The conversion table can be found by studying the code (getLiPoBatteryLevel()).

You can't directly read the battery voltage, but you can use an ADC (Analog to Digital Converted) to get a value within a known interval. An ADC can only read voltages in certain ranges, so if the read voltage is higher than the ADC range, you must scale it down to fit than range using a voltage divider.

ESP8266 has a single 10bit ADC pin. 1 Volt is the max input to the internal ADC and will give a Raw reading of 1023. Wemos D1 Mini already contains a voltage divider on the ADC pin. This is a 220 k resistor over a 100 k resistor. As long as the voltage is bellow 3.3V you could directly connect the battery to the exposed ADC pin.



Wemos D1 Mini Internal voltage divider

But 4.2V of a fully charged battery is more than the exposed pin can handle,. By connecting an additional 220k resistor from battery positive terminal to the ADC pin, we will get less than 3.3V on the exposed pin, and less than 1V, on the ADC pin of ESP8266 because of the existing voltage divider.

By adding a 220k resistor, it will in fact be a total resistance of 220k+220k+100k=540k. So if the Voltage of the fully charged battery would be 4.2 Volt, the ADC of the ESP8266 would get 4.2 * 100/540= 0.77 Volt, allowing for an input voltage up to 5.4V when ADC sees 1V, so the input pin and ADC will be protected in case of over-voltage (direct USB voltage applied).

I added a ceramic capacitor of 100nF between the ADC pin and ground to smooth out the readings and reduce noise. I just take 1 sample at sketch start and multiply it by a known calibration factor, as the reading were very stable. If your readings vary, you might try a larger capacitor, reading multiple samples and averaging. But if you have noise you should first check the connections and make sure you have short wires (like 10cm) to minimize inductance.

Nokia 5110 Display

These old displays are still out there. The Nokia 5110 LCD is a 84 x 48 pixels Monochrome display, it’s cheap and can be easily used with the Adafruit GFX Library. It lets you display text, shapes and monochrome images. It also features a led back-light. It consumes just 6-7 mA without the back-light.

The LCD needs to be powered from 2.7 - 3.3V. The back-light consists of 4 leds. I could’t get those leds consume more than 20mA in my module. It could have a limiting resistor or they could just have a higher forward voltage. Some modules state that they can use up to 100mA and that’s really possible as a typical white led can consume 30mA, so I wouldn’t be surprised to see 120mA consumed by the backlight. Compared to the 6-7mA needed by the display,

this is something you should be aware of. That’s why on battery operated devices I would turn the back-light on only when needed.

I saw some modules that require logic LOW for the LED BLT pin. That’s why you should always read the datasheet and if it directly needs a power supply always put a limiting resistor. I used a 22ohm resistor for example just to be safe.

Built into this LCD is a Philips PCD8544 display controller, which converts the massive parallel interface of the raw LCD to a more convenient serial one. The PCD8544 is controlled through a synchronous serial interface similar to SPI. There are clock (**SCLK**) and data (**DN**) input lines, and an active-low chip select (**SCE**) input as well.

On top of those three serial lines, there is another input – **D/C** – which tells the display whether the data it’s receiving is a command or displayable data.

For the data transmission pins – SCLK and DN(MOSI) – I used the Arduino’s **hardware SPI pins**, which will help to achieve a faster data transfer. The chip select (SCE) is connected to ground, reset (RST) is connected to Wemos R1 Mini reset pin through a 10K resistor,, in order to reset the display when the micro-controller resets. The data/command (D/C) pin can be connected to **any digital I/O pin**, so I used **GPIO12** normally intended for SPI MISO, which is unused, labeled **D6** on the D1 Mini.

Sparkfun created a nice tutorial on using this LCD.

Auto LCD Back-light

Since the back-light LCDs can be power hungry and are not always needed, I created a cirtuit to adjust the back-light according to the ligth conditions.

The auto back-light circuit I used consists out of a NPN transistor , a LDR (Light Dependent Resistor) and two resistors. The NPN transistor used is **2N3904**, which can provide 200mA. Since, the required current is low, you have many options to replace this with any NPN transistor. In this setup, the NPN is not used as it's normally used as a low side switch, but as a high side switch. Using it on the low side (feed negative) wouldn't have been possible since the BLT pin requires a positive voltage to turn on, as the ground is already connected on the module

Without the LDR, the circuit makes the back-light fully lit. With the LDR, the transistor base will get more or less current depending on how much light will the LDR get, because the LDR resistance decreases with the light from very high resistance in dark to very low resistance in light. The transistor base will get a very low current in light environments, because most of the current will travel through the less resistive path of the LDR, so the current from collector

to emitter (3.3V to BLT pin) will be low and the back-light will be very dimmed or off. In darker environments, the LDR resistance increases, allowing more current to reach to the base of the transistor, which allows more current to travel from collector to emitter (3.3V to BLT pin) and light up the leds.

I chose a fixed 1K resistor from 3.3V to transistor base, and a protection resistor of 22ohms from emitter to BLT. The protection is not really necessary but a resistor is always recommended before a led.

Logo LED

Every product needs a logo, and what's fancier than a light up logo?

For this to work, I just used **GPIO 0** labeled as **D3** on *Wemos D1 Mini*, set it HIGH when the logo should be lit, and LOW when the logo should be turned off. I also used PWM with analogWrite to dim it out. Of course, I added a protection resistor of a few ohms (I actually used 1.2ohm, but I recommend a larger one if you are not sure of the led forward voltage like 22ohm) to keep the led safe.

GPIO 0 is a special pin of ESP8266, and is used in the boot process, so it's always going to be HIGH at boot, but this was perfectly fine. After boot, you can use it as output and do whatever you like with it.

Libraries

•WiFi Manager - used for the network configuration portal
•Adafruit GFX - Graphics library
•Adafruit PCD8544. - Library for the internal chip of the Nokia 5110 LCD. Info for using it with Wemos D1 Mini here.
•Pull Request for ESP8266 support into Adafruit-PCD8544-Nokia-5110-LCD-library (not merged at the moment I am writing this. If this has not been merged yet, you will need to manually copy the changes across).
•SimpleTimer - used for non-blocking timers
•Double reset detector - detects double presses on the chip's reset button
•MovingAverage - algorithm to smooth out the scale readings.

API Reference

The restful HTTP API is available at `http://<scale-IP/api/`

The API receives the input parameters as form-encoded. Example with curl:

```
curl -XPATCH /api/device/settings --data
"useStaticIP=1&ip=192.168.1.11"
```

Successful POST/PATCH actions return HTTP status code 200 and a JSON like this:

```
{"success": true}
```

Failed requests return a non-200 HTTP status code and a JSON object with two keys:

•error: *bool* (always false)

•message: *string*

Example:

```
{"error": "true", "message": "Invalid voltage. Must be between 3 - 4.35v"}
```

All paths bellow are relative to the scale IP address.

Device API

Retrieve info

`GET /api/device/info`

Sample response


```
{
    "battery": {
        "voltage": 4.09,
        "chargeLevel": 90
    },
    "network": {
        "deviceMac": "84:F3:EB:B3:86:75",
        "ip": "192.168.1.11",
        "subnetMask": "255.255.255.0",
        "gateway": "192.168.1.1",
        "dnsIP": "192.168.1.1",
        "SSID": "tplink-est",
        "BSSID": "64:66:B3:64:1D:A9",
        "signalStrength": -71
    }
}
```


Calibrate ADC

`POST /api/device/calibrate-adc`

Calibrate the internal ADC for reading the battery voltage. For this to work you will send the measured battery voltage as parameter.

Required params:

•voltage *float* Value must be between 3 and 4.35

Power off

POST/api/device/poweroff

Reset

POST/api/device/reset

Make sure you keep the power-on button pressed while doing a soft-reset,otherwise it will kill it’s own power during reset. This only works without pressing the button if you are debugging the scale with an USB cable plugged into Wemos D1 Mini.

Settings API

Retrieve settings

GET /api/settings

Sample response:

```
{
    "useStaticIp": true,
    "ip": "192.168.1.11",
    "gateway": "192.168.1.1",
    "subnetMask": "255.255.255.0",
    "zeroFactor": 131702,
    "calibrationFactor": 414.19,
    "calibrationWeight": 152.00,
    "powerOffTimerSec": 300,
    "idlePowerOffTimerSec": 90,
    "voltageCalibrationFactor": 0.005226
}
```

Change settings

PATCH /api/settings

Sample request with curl:

```
$ curl -XPATCH http://192.168.1.11/api/settings --data
"useStaticIp=1&ip=192.168.0.10"
```

Scale functions API

Calibrate scale

POST /api/calibrate

Starts the interactive calibration mode. The scale will ask you to place a known weight then it will auto-calibrate.

Retrieve weight for last set food id

`GET /api/weight`

Response:

```
{
   “foodId”:123,
   “weight”:90.5,
   “unit”:“g”
}
```

Set food info

`POST/api/weight`

Required params:

•foodId string an unique id to identify this food
•name string food name for display on the scale LCD
•calories float caloriers/100g
Sample request:

```
curl -XPOST http://192.168.1.11/api/weight --data
"foodId=123&name=Banana&calories=89.2"
```

Tare

`POST /api/tare`