HeartBeats - Smart Music Player

by Ibe Verbeke

Here I would like to walk you through the process of making a music player that picks music based on your own heartbeat, the working title for my take on this project was HeartBeats and I ultimately kept it as a final name.

HeartBeats is a raspberry Pi Project I made for school, the assignment was about connecting different sensors to a Raspberry Pi and processing their measurements in a python program, we also made a corresponding website and a database to save all of the incoming data about the project, so that will also be included in this Instructable!

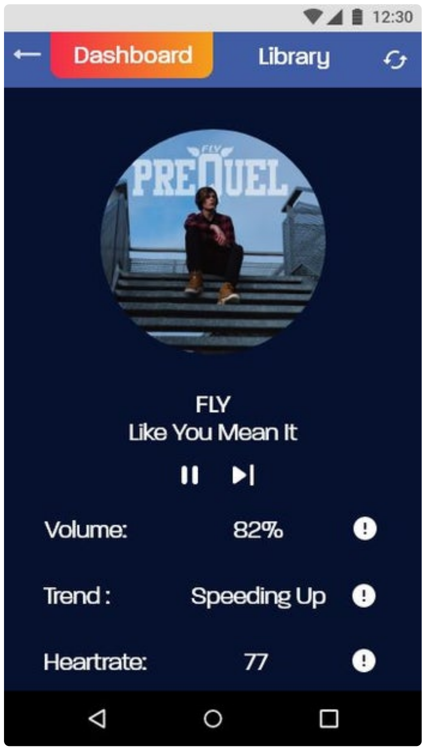
Let's get started:

Supplies:

The idea behind this project is quite simple but it does require a decent amount of supplies and preparation:

You will need:

- A Raspberry Pi
- A Micro-SD card (16GB)
- A 16x2 LCD Display (or any 8-bit LCD display for that matter)
- An dafruit T cobbler
- An MCP3008 chip
- A Pulse sensor
- A Thermistor (I used a 47K ohm thermistor)
- 10K Ohm resistor
- 10K Ohm rotary potentiometer
- 10K Ohm trimmer
- A push button switch
- A 220 Ohm resistor
- breadboard (if you are planning on buying a smallish breadboard I recommend buying 2, you'll use quite some space on it)
- 1x 5v breadboard puwer supply
- A speaker with an AUX connection
- An AUX cord
- A container of your choice (this is not mandatory but it looks way better in my opinion)
- A bunch of jumper wires
- A couple pieces of Velcro



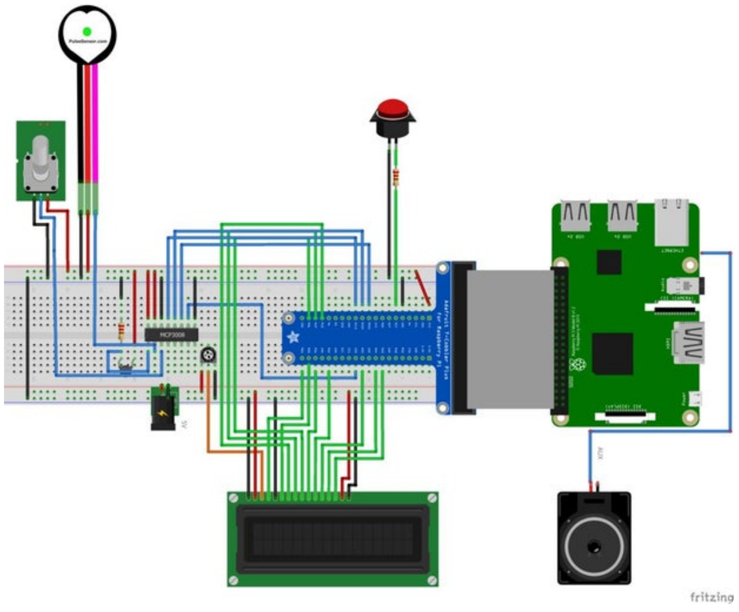
Step 1: Preparation

I highly recommend taking a minute to think about how you personally would like to approach this project.

If you just plan on making my exact version, feel free to skip this step, but the real beauty in this project is really in making it your own.

Step 2: Circuit

In the photo above, you can already take a look at the circuit we are working towards! Don't stress yet, I will walk you through all the important bits step by step. This is a good time to check if you have all the necessary components and enough breadboard space to get everything to work.



Step 3: Getting Your Pi Ready

All of the stuff we're about to attach to the Raspberry Pi will first pass through a T-cobbler piece that we can plug into our breadboard. Make sure that the orientation of your T-cobbler and Raspberry Pi is correct (shown in picture) before you plug everything in. You really don't wanna make a mistake here as it can permanently damage the pins on your Pi.

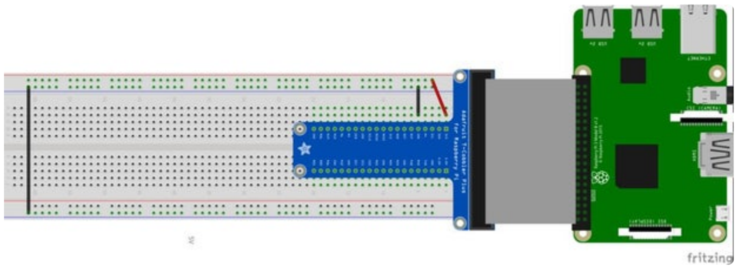
A second thing you can already do is attach the 3.3V pin of the T-cobbler to the red lane of the breadboard, and attach the GND pin to the blue lane. It's also best practice to already link the 2 blue lanes to eachother (left), for now it won't do much, but that will get important later.

Tips:

- I recommend using a consistent color code for wires, the most standard would be : + : red / - : black, in my case I went for + : green / - : White as this were the wires I had at my disposal. later we will also use a third color for signal wires.

- Never change anything to a circuit while the Pi is plugged in to the 5V plug

I also recommend jumping ahead to the coding section of this tutorial to configure your Raspberry Pi correctly

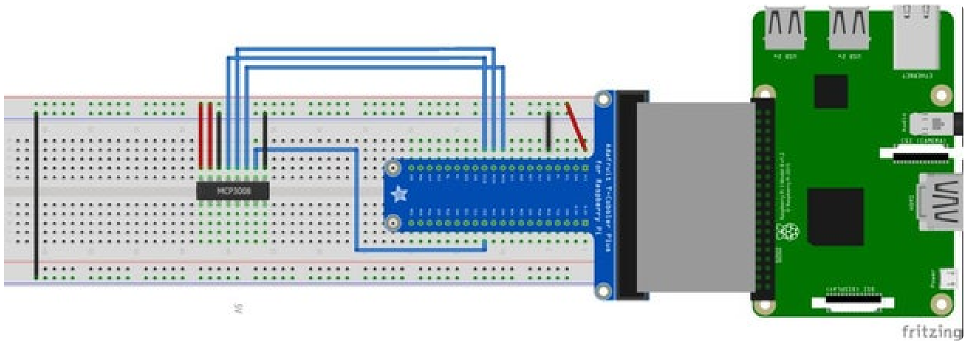


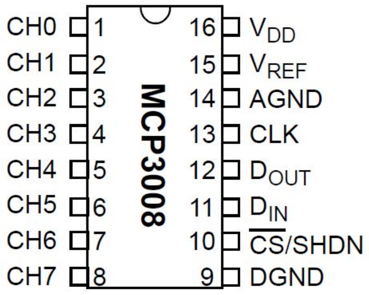
Step 4: MCP 3008

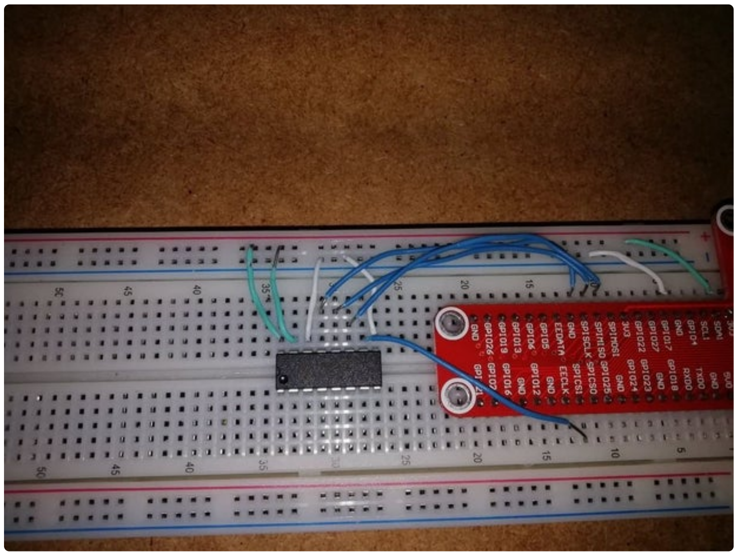
The MCP 3008 chip is used to read analog signals, and will be used to read all of our sensors. So it's logical that we attach this to the breadboard first.

Make sure that the orientation of the MCP 3008 chip is correct (look at the little nudge it has on one side) you don't want to put 3.3V on a bare channel pin!

Right now we don't have a real way of testing if our MCP chip works correctly, but we will go over that once we have attached a sensor.







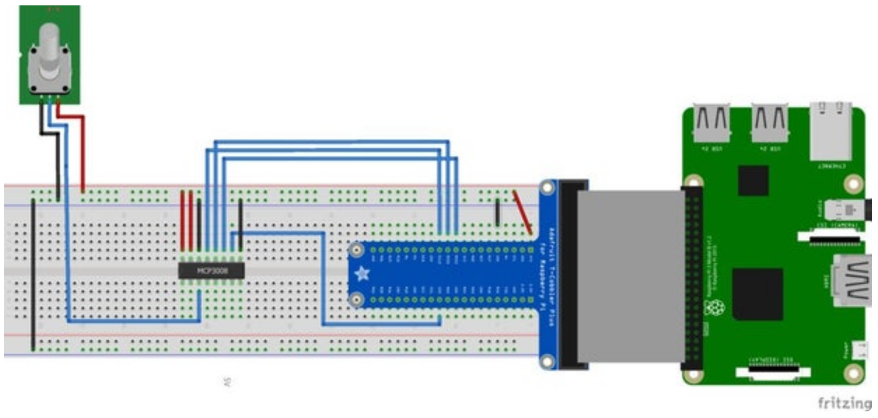
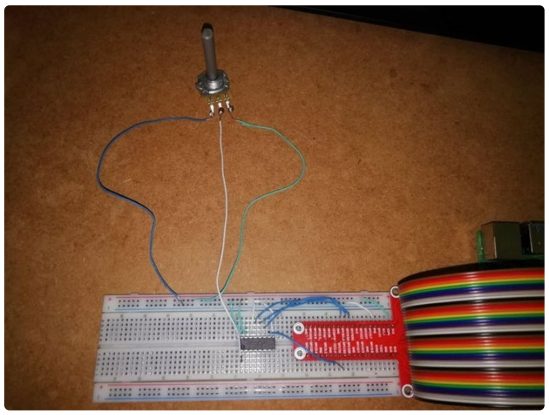
Step 5: Volume Measurement: Potentiometer

The first sensor we are going to attach is the Potentiometer, we'll use this sensor to control the output volume of our music player.

Attach one of the outer pins to the 3.3V (red lane) out, and the other to the GND (blue lane). The middle pin is used to measure the difference in resistance, and is the signal that we actually need to read! Attach this pin to the CH0 pin (most left in the pictures) of the MCP 3008.

Before we attach all the other stuff, I recommend skipping ahead to the coding part of this tutorial to learn more about testing all the sensors.

Once you've tested the functionality of the potentiometer we can get to the Temperature Sensor.

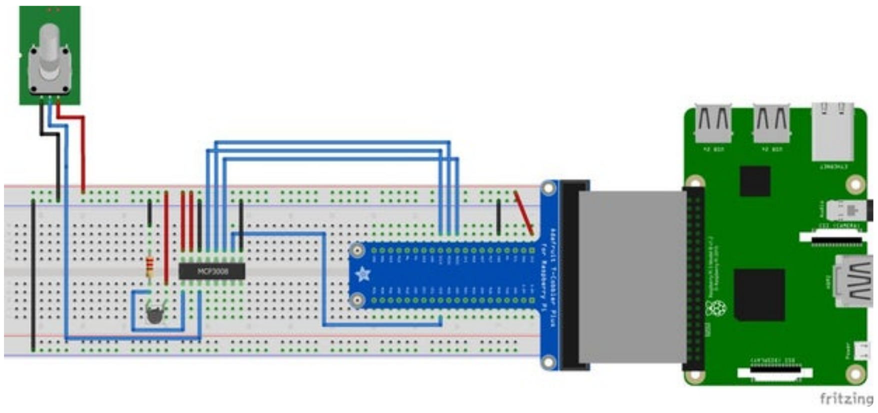
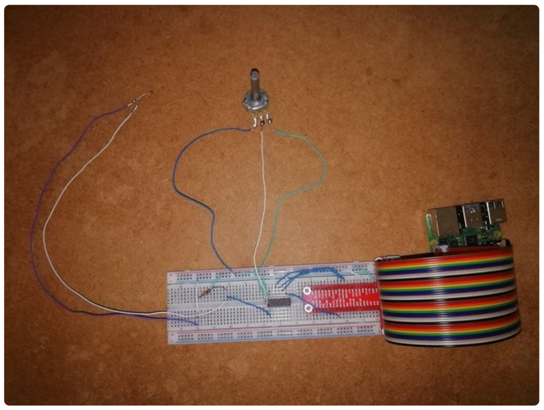
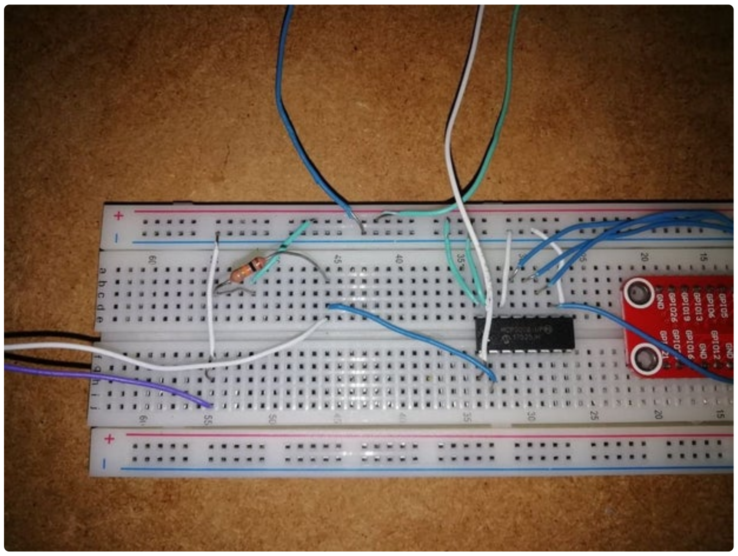
Step 6: Temperature Measurement: Thermistor

This part of the circuit can look a bit messy, but is actually quite simple!

Basically you want to connect the 10K Ohm resistor and the Thermistor to eachother and measure the signal in between both.

The signal wire that we connected in between will be pluged in at the CH1 pin of the MCP3008

Once again: test this sensor at the coding section of this tutorial!

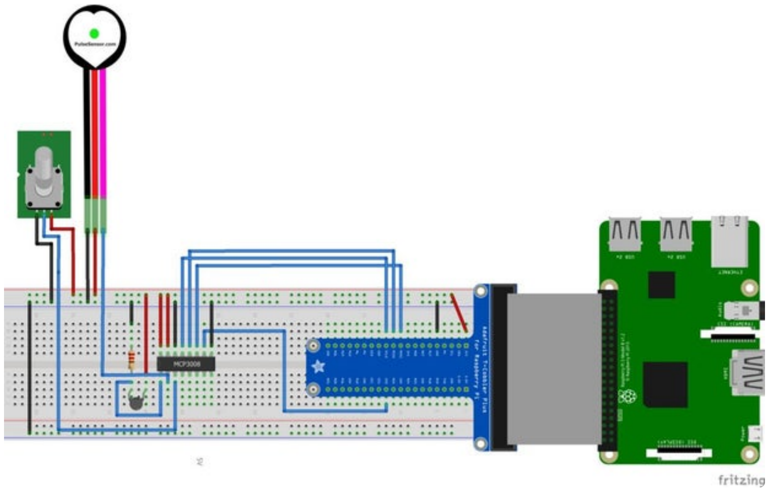




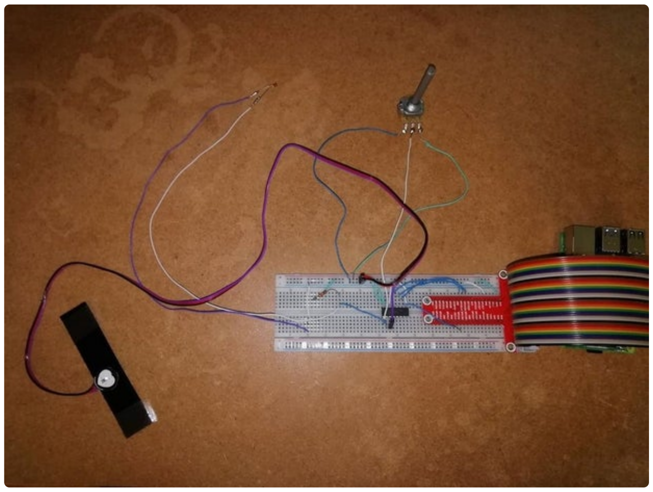
Step 7: Pulse Sensor

If you attached your Thermistor correctly, this next sensor should be kids' stuff!

The red and black wires are traditionally the positive and negative poles of the sensor, so these should be attached to the 3.3V Out (red lane) and the GND (blue lane) respectively. The purple wire is the pulse signal, so we will attach this one to the CH2 pin of the MCP3008

Let's proceed!



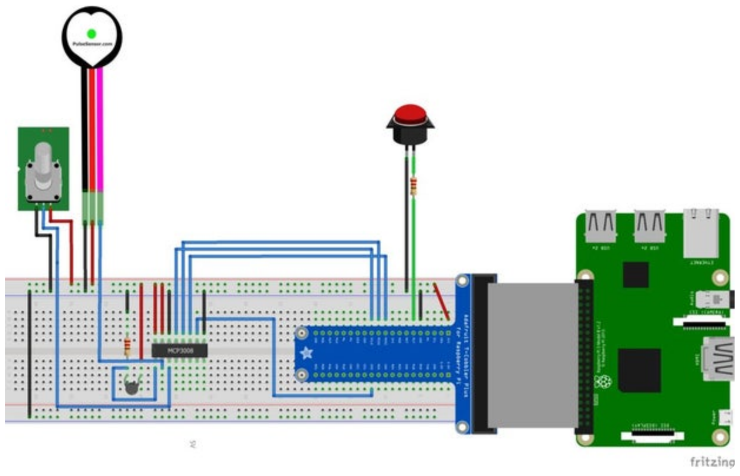


Step 8: Start Button

We'll also need a button to actually pick and start a song after we've measured all the different parameters.

Connect the Anode pin of the button to the GPIO #17 pin and the Cathode pin to the GND (blue lane). I also recommend protecting the button (and your Pi pins) by adding a 220 Ohm resistor in the circuit.

Now you should skip ahead to the coding section and test out the button!



Step 9: LCD Display

The most challenging part of the circuit is probably the LCD Display. I really recommend taking your time and not taking too many risks for this one! Do not power up your Pi if you're not sure that all the pins are connected correctly. Read all the pin names carefully, and doublecheck all your connections before proceeding!

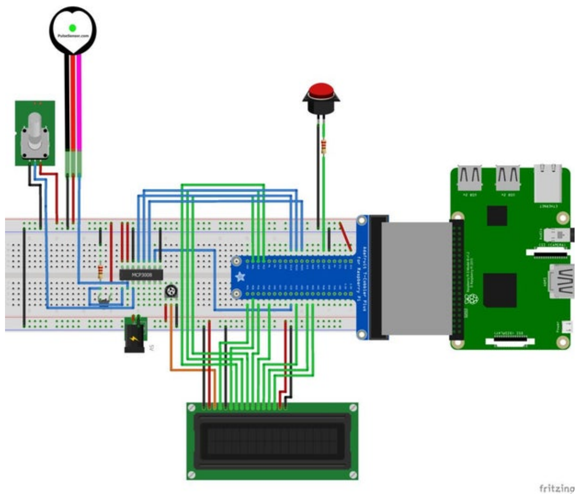
I chose to use a second breadboard to forward all the connections, this way of working provided a better overview of my circuit and made it easier to test everything and catch errors.

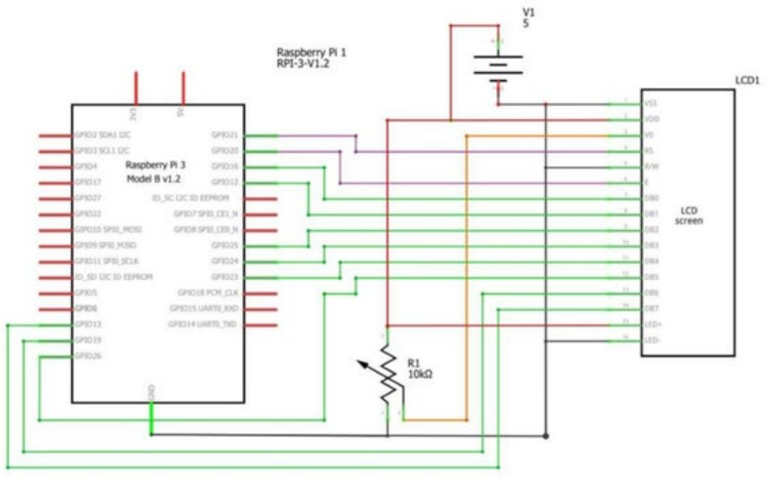
This is also where the extra 5V breadboard power supply comes in, attach it to (ideally) your second breadboard and make sure the GND (blue lane) is connected to the other GND lane of the breadboard (the one from the T-cobbler).

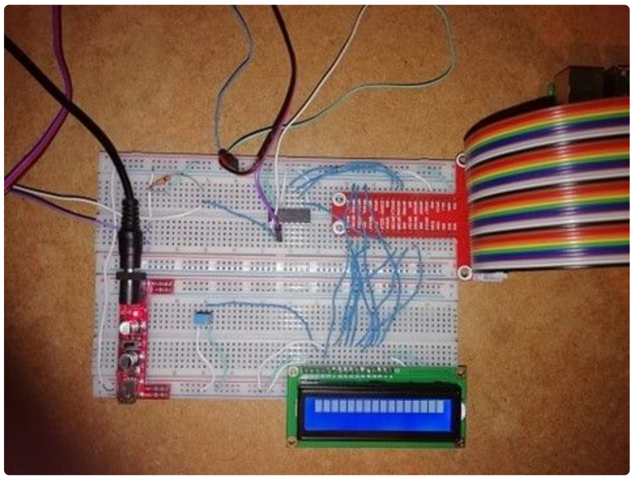
Don't forget to connect a Trimmer to the RW pin of the LCD Display, otherwise you probably won't see a lot on the screen once you test it.

Once everything is connected and tested, we should be done connecting stuff for a while!

Let's get to the coding part







Step 10: Getting Your Pi Ready for Use (software)

First of all we're going to install an image file to the SD card, this is the Operating system that the Pi will use to function.

The image for this project can be found right here:

If you don't know how to write a Raspberry Pi image to an SD card, I suggest you google that exact sentence, there are far better tutorials on this than I could ever explain in this Instructable.

Once the image is written to the SD card, plug it into the Pi and power it up. You can also already plug in the LAN cable into your Pi, with the other end attached to your computer.

Open up the commandprompt on your computer and type this phrase: ping 169.254.10.1

If you get a reply from that ping action, you are ready to proceed to the next step!

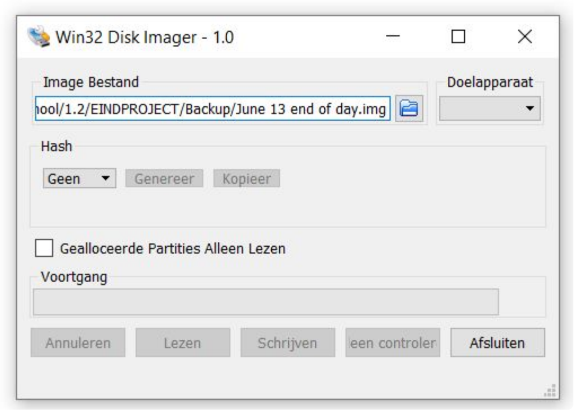
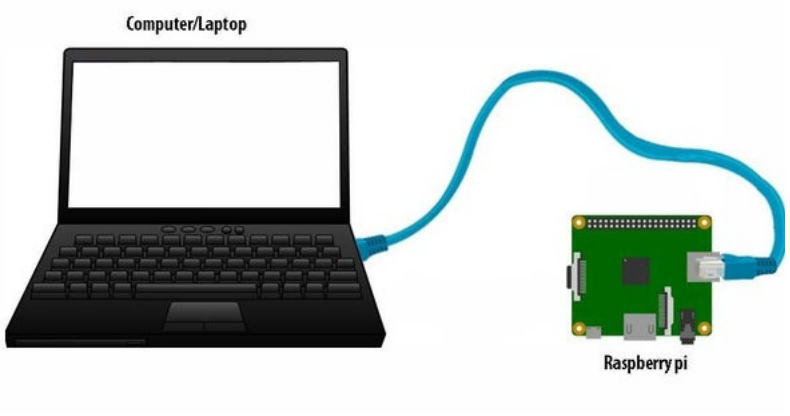
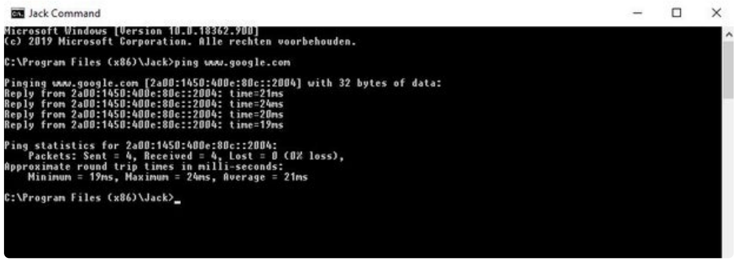


Image Bestand: ool/1.2/EINDPROJECT/Backup/June 13 end of day.img





```
Microsoft Windows [Version 10.0.18362.900]
(c) 2019 Microsoft Corporation. Alle rechten voorbehouden.

C:\Program Files (x86)\Jack>ping www.google.com

Pinging www.google.com [2a00:1450:400e:80c::2004] with 32 bytes of data:
Reply from 2a00:1450:400e:80c::2004: time=21ms
Reply from 2a00:1450:400e:80c::2004: time=24ms
Reply from 2a00:1450:400e:80c::2004: time=20ms
Reply from 2a00:1450:400e:80c::2004: time=19ms

Ping statistics for 2a00:1450:400e:80c::2004:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 19ms, Maximum = 24ms, Average = 21ms

C:\Program Files (x86)\Jack>_
```

Step 11: MySQL Workbench

MySQL workbench is the program that we will use to configure the database of the project

It can be downloaded here: https://www.mysql.com/products/workbench/

Install it on your computer and boot it up!

While the Pi is powered up and attached to your computer: Configure a new connection in MySQL workbench with the settings seen in the first picture.

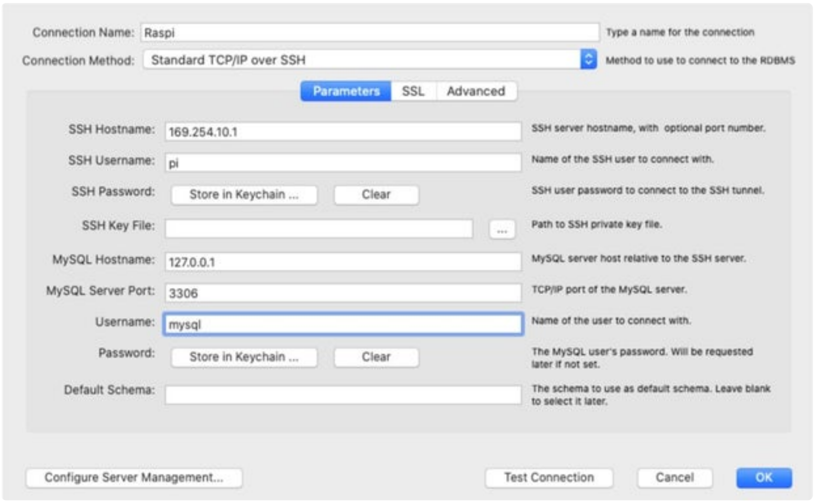
The password for this is the standard : mysql (same as the username) we will change this later.

You might get a warning after you hit OK, in that case press 'continue anyway'.

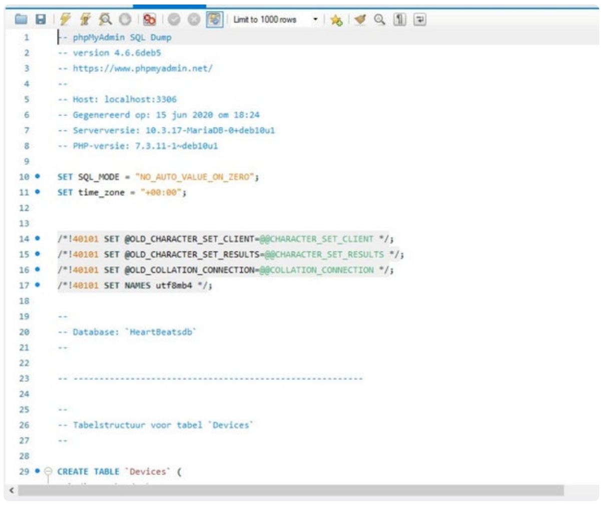
Now you should be able to open a connection to the database system of your Pi.

As a last step we are going to configure our database in the system. Choose File > Open SQL script, and locate the HeartBeats.sql file. Once it's loaded in hit the little lightning icon to apply the scripted code.

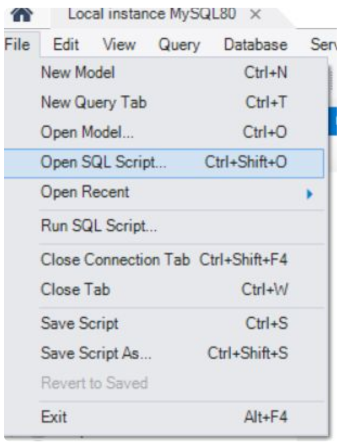
If you don't get any errors now, your database should be fully functional!

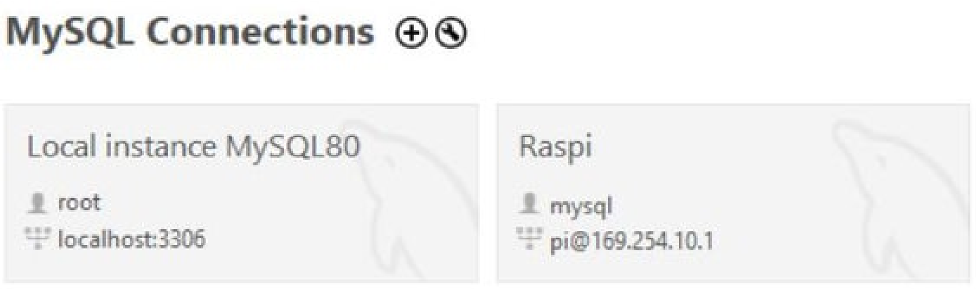


Connection Name:	Raspi	Type a name for the connection
Connection Method:	Standard TCP/IP over SSH	Method to use to connect to the RDBMS
SSH Hostname:	169.254.10.1	SSH server hostname, with optional port number.
SSH Username:	pi	Name of the SSH user to connect with.
SSH Password:	Store in Keychain ... Clear	SSH user password to connect to the SSH tunnel.
SSH Key File:		Path to SSH private key file.
MySQL Hostname:	127.0.0.1	MySQL server host relative to the SSH server.
MySQL Server Port:	3306	TCP/IP port of the MySQL server.
Username:	mysql	Name of the user to connect with.
Password:	Store in Keychain ... Clear	The MySQL user's password. Will be requested later if not set.
Default Schema:		The schema to use as default schema. Leave blank to select it later.



```
-- phpMyAdmin SQL Dump
-- version 4.6.6deb5
-- https://www.phpmyadmin.net/
--
-- Host: localhost:3306
-- Gegenereerd op: 15 jun 2020 om 18:24
-- Serverversie: 10.3.17-MariaDB-0+deb10u1
-- PHP-versie: 7.3.11-1~deb10u1

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `HeartBeatsdb`
--

-- --------------------------------------------------------

--
-- Tabelstructuur voor tabel `Devices`
--

CREATE TABLE `Devices` (
```





Step 12: Visual Studio Code: SSH Setup

As our IDE, we will be using visual studio code!

You can download and install it here: https://code.visualstudio.com/

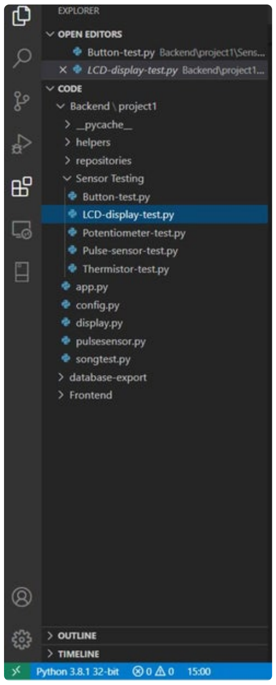
Once installed, click the Extensions icon in the left menu and search for SSH, install the SSH - Remote extension! This will set up a connection so we can code directly onto our Pi

Once installed you should see 2 small green arrows at the left bottom of your screen, click 'em! This should give you a little popup to set up your SSH connection. Choose Add new SSH host.

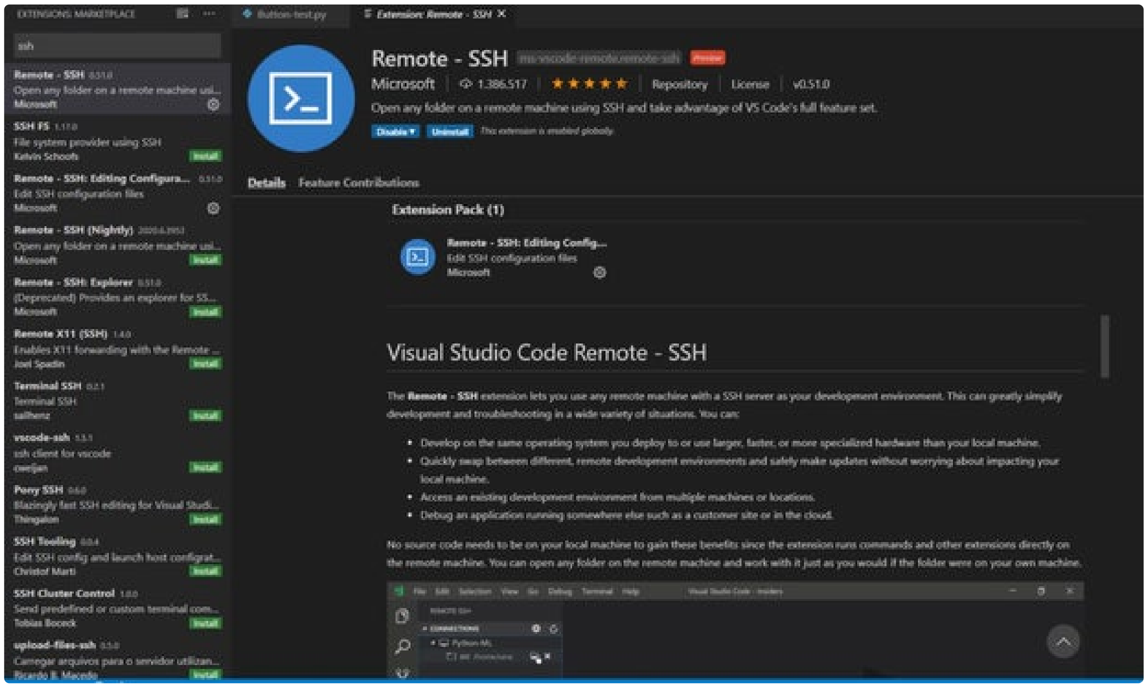
Fill in the following statement: ssh pi@169.254.10.1 -A

Now you should get a password prompt: the password for a fresh OS is: raspberry

Now you should have a succesful SSH connection to your Raspberry Pi!



```
EXPLORER
∨ OPEN EDITORS
    Button-test.py Backend\project1\Sens...
  × LCD-display-test.py Backend\project1...
∨ CODE
  ∨ Backend \ project1
    > __pycache__
    > helpers
    > repositories
    ∨ Sensor Testing
        Button-test.py
        LCD-display-test.py
        Potentiometer-test.py
        Pulse-sensor-test.py
        Thermistor-test.py
      app.py
      config.py
      display.py
      pulsesensor.py
      songtest.py
  > database-export
  > Frontend
> OUTLINE
> TIMELINE
Python 3.8.1 32-bit  ⊗0 ⚠0  15:00
```



```
EXTENSIONS: MARKETPLACE
ssh
Remote - SSH 0.51.0
Open any folder on a remote machine usi...
Microsoft
SSH FS 1.17.0
File system provider using SSH
Kelvin Schoofs                      Install
Remote - SSH: Editing Configura... 0.51.0
Edit SSH configuration files
Microsoft
Remote - SSH (Nightly)
Open any folder on a remote machine usi...
Microsoft                           Install
Remote - SSH: Explorer 0.51.0
(Deprecated) Provides an explorer for SS...
Microsoft                           Install
Remote X11 (SSH) 1.4.0
Enables X11 forwarding with the Remote ...
Joel Spadin                         Install
Terminal SSH 0.2.1
Terminal SSH
sailhenz                            Install
vscode-ssh 1.5.1
ssh client for vscode
cweijan                             Install
Pony SSH 0.6.0
Blazingly fast SSH editing for Visual Studi...
Thingalon                           Install
SSH Tooling 0.0.4
Edit SSH config and launch host configurat...
Christof Marti                      Install
SSH Cluster Control 1.0.0
Send predefined or custom terminal com...
Tobias Boceck                       Install
upload-files-ssh 0.5.0
Carregar arquivos para o servidor utilizan...
Ricardo B. Macedo                   Install
```

Remote - SSH

Microsoft | 1.386.517 | ★★★★★ | Repository | License | v0.51.0

Open any folder on a remote machine using SSH and take advantage of VS Code's full feature set.

Disable ▾ Uninstall This extension is enabled globally.

Details Feature Contributions

Extension Pack (1)

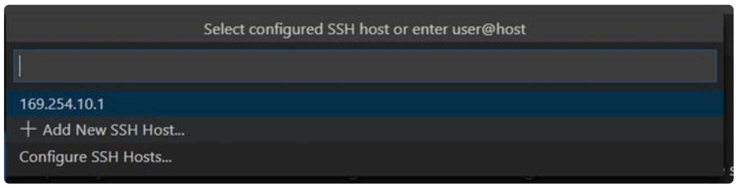
Remote - SSH: Editing Config...
Edit SSH configuration files
Microsoft

Visual Studio Code Remote - SSH

The **Remote - SSH** extension lets you use any remote machine with a SSH server as your development environment. This can greatly simplify development and troubleshooting in a wide variety of situations. You can:

- Develop on the same operating system you deploy to or use larger, faster, or more specialized hardware than your local machine.
- Quickly swap between different, remote development environments and safely make updates without worrying about impacting your local machine.
- Access an existing development environment from multiple machines or locations.
- Debug an application running somewhere else such as a customer site or in the cloud.

No source code needs to be on your local machine to gain these benefits since the extension runs commands and other extensions directly on the remote machine. You can open any folder on the remote machine and work with it just as you would if the folder were on your own machine.



```
Select configured SSH host or enter user@host
|
169.254.10.1
+ Add New SSH Host...
Configure SSH Hosts...
```

Step 13: Time to Really Start

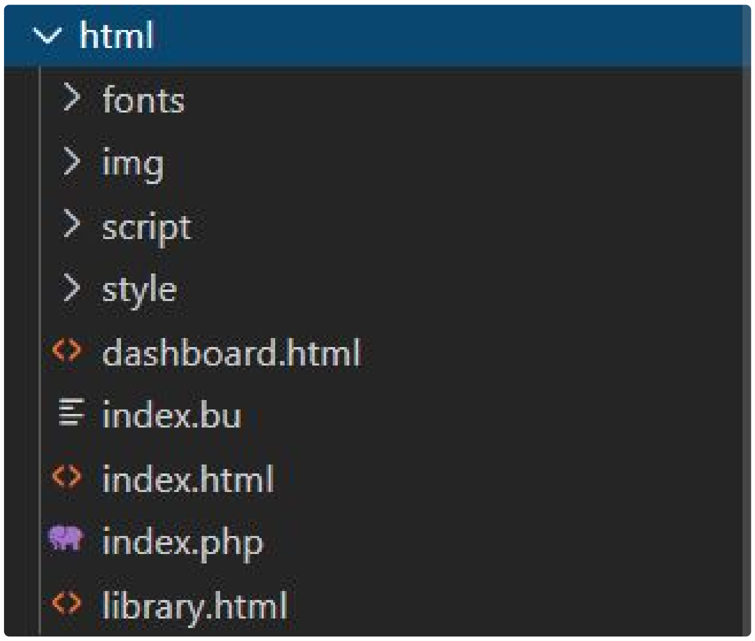
Now we can start testing! Or at least... almost

First we have to check if all the code is in the right directory.

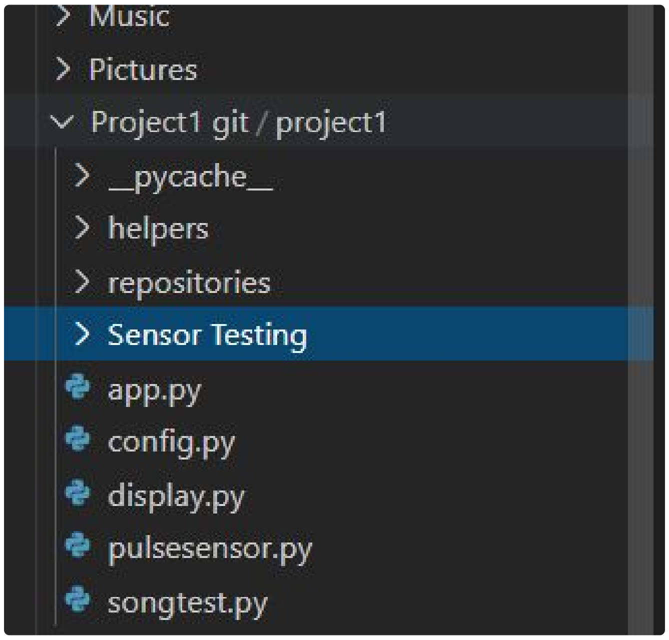
all the python files should be in the project1 directory, do not change the structure of these folders! You might just break one of the scripts by doing so.

The frontend code should be inside the var/www/html directory. Here it's also wise to not change the file structure inside.

The image should also come with 50 songs already in the music folder! You're welcome



```
html
  fonts
  img
  script
  style
  dashboard.html
  index.bu
  index.html
  index.php
  library.html
```

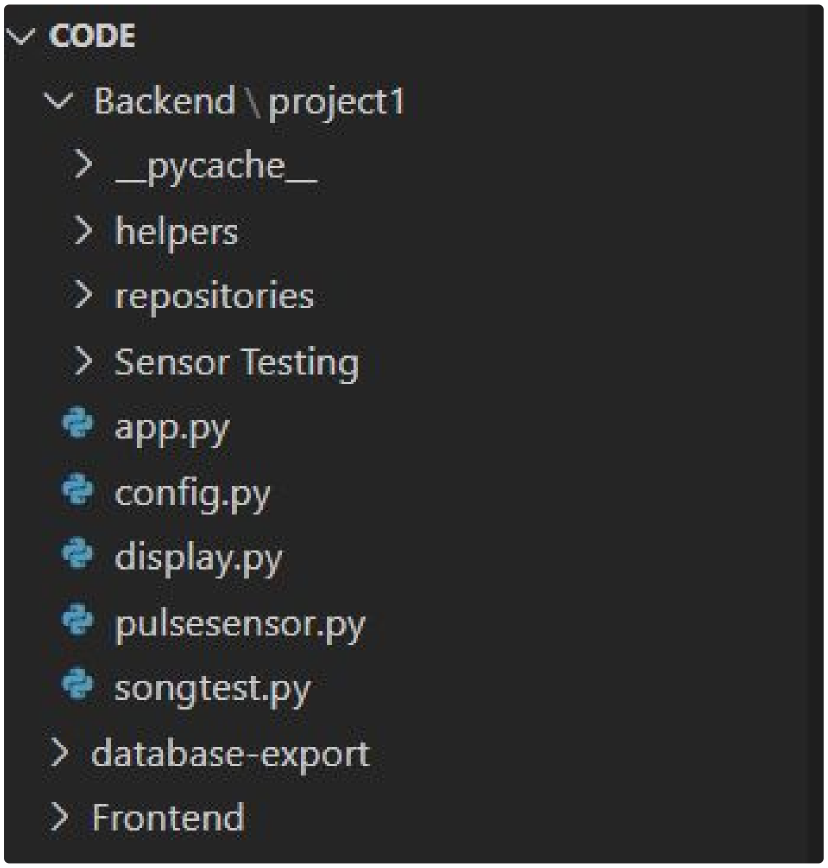


```
Music
Pictures
Project1 git / project1
  __pycache__
  helpers
  repositories
  Sensor Testing
  app.py
  config.py
  display.py
  pulsesensor.py
  songtest.py
```

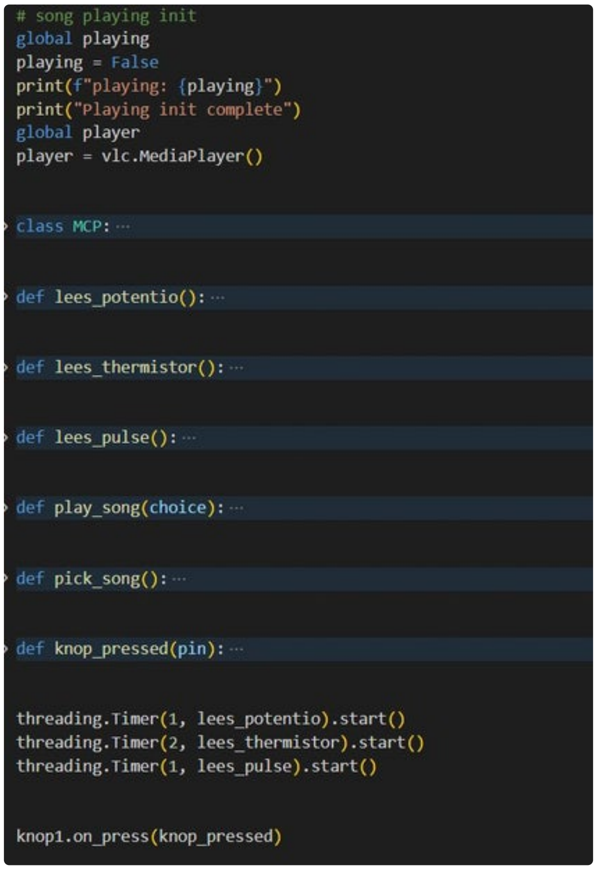
Step 14: The Code

Most of the code used for this project was written in a single python script. But I also included some split up scripts so we can test out all of our components separately.

Let's first do that



```
CODE
  Backend \ project1
    __pycache__
    helpers
    repositories
    Sensor Testing
    app.py
    config.py
    display.py
    pulsesensor.py
    songtest.py
  database-export
  Frontend
```



```
# song playing init
global playing
playing = False
print(f"playing: {playing}")
print("Playing init complete")
global player
player = vlc.MediaPlayer()


class MCP:···


def lees_potentio():···


def lees_thermistor():···


def lees_pulse():···


def play_song(choice):···


def pick_song():···


def knop_pressed(pin):···


threading.Timer(1, lees_potentio).start()
threading.Timer(2, lees_thermistor).start()
threading.Timer(1, lees_pulse).start()


knop1.on_press(knop_pressed)
```

Step 15: Testing: Potentiometer

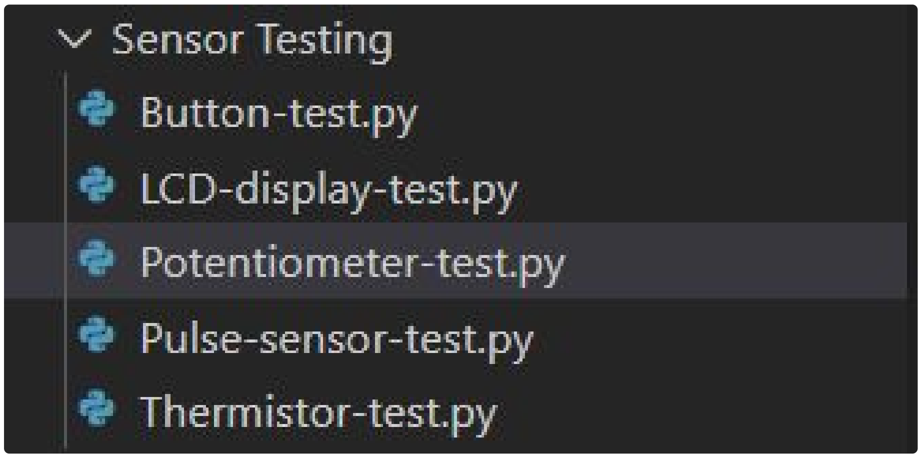
In this step we are going to test our Potentiometer, but also our MCP3008, since this is the first sensor that we are connecting.

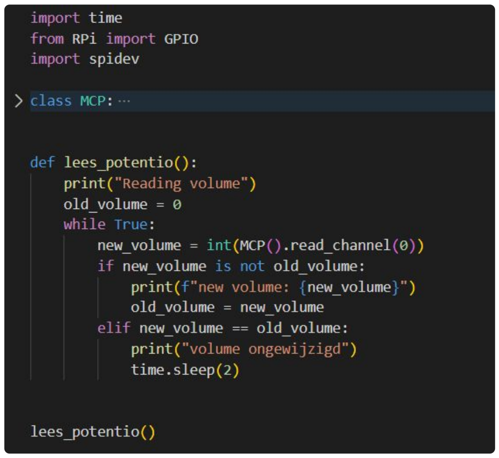
I included some scripts to test out all the different sensors, for this one specific you should run : Potentiometer-test.py within the backend/sensor-testing folder

Running A script is simply done by hitting the small green play button in the upper right corner

If you have done everything correctly up to this point, you should receive a string in the terminal saying "new volume: *a value between 0 - 100*

If you consistently get a 0 or get no value at all, even while dialing the knob, I suggest you check all the previous steps for errors!





```
import time
from RPi import GPIO
import spidev

> class MCP:···


def lees_potentio():
    print("Reading volume")
    old_volume = 0
    while True:
        new_volume = int(MCP().read_channel(0))
        if new_volume is not old_volume:
            print(f"new volume: {new_volume}")
            old_volume = new_volume
        elif new_volume == old_volume:
            print("volume ongewijzigd")
            time.sleep(2)


lees_potentio()
```

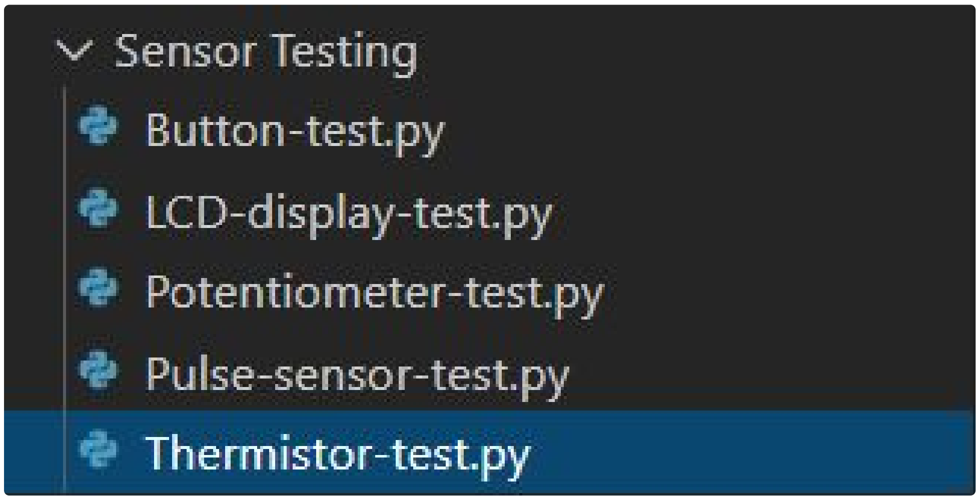
Step 16: Testing: Thermistor

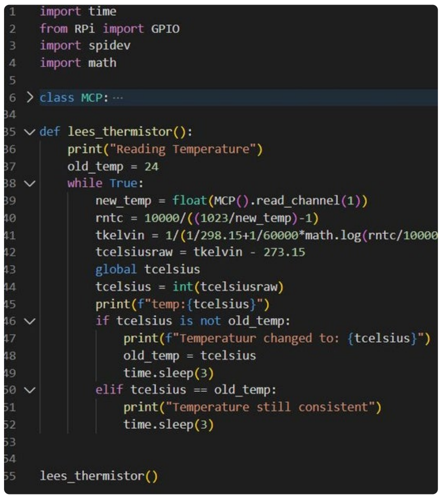
For this test: run Thermistor-test.py

You should get a string printed in the terminal every 3-4 seconds

The string can either say: "temp: *a realistic value for your environment*" or "Temperature changed to: *value* or "Temperature still consistent".

Once you get a consistent temperature I suggest putting your index finger on the sensor, this should genuinly give an increase in temperature (given that you don't live in a tropical forest or desert).





```
import time
from RPi import GPIO
import spidev
import math

class MCP:···

def lees_thermistor():
    print("Reading Temperature")
    old_temp = 24
    while True:
        new_temp = float(MCP().read_channel(1))
        rntc = 10000/((1023/new_temp)-1)
        tkelvin = 1/(1/298.15+1/60000*math.log(rntc/10000
        tcelsiusraw = tkelvin - 273.15
        global tcelsius
        tcelsius = int(tcelsiusraw)
        print(f"temp:{tcelsius}")
        if tcelsius is not old_temp:
            print(f"Temperatuur changed to: {tcelsius}")
            old_temp = tcelsius
            time.sleep(3)
        elif tcelsius == old_temp:
            print("Temperature still consistent")
            time.sleep(3)


lees_thermistor()
```

Step 17: Testing: Pulse Sensor

Testing and callibrating the Pulse sensor can be a little tricky.

First off load up the Pulse-sensor-test.py script

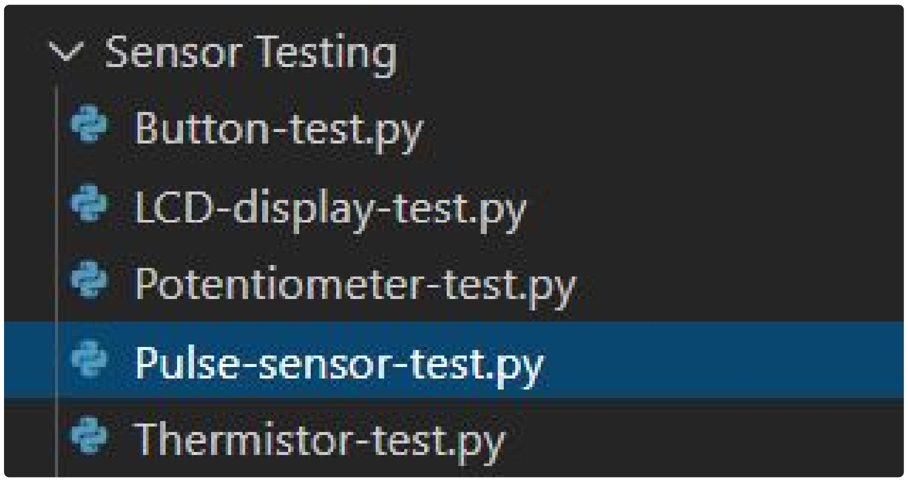
When you start running the script immediately put your finger on the sensor, this will give the most consistent and correct results for starters.

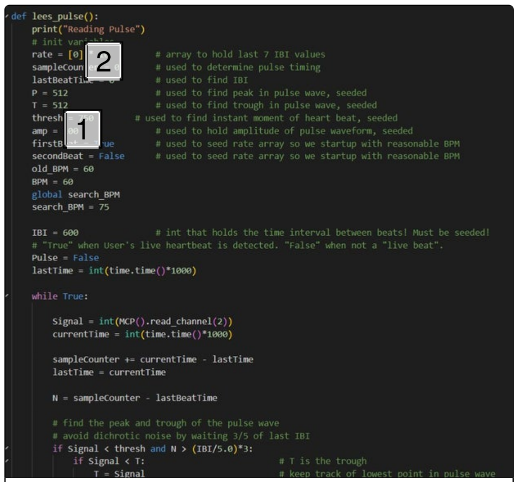
In the terminal you should see a certain heartrate (the first few heartrates might seem extreme, this is because the script is still calculating it).

If you are not happy with how the sensor reacts, there are 2 main parameters you can change:

- If the sensor seems to react way too often, giving you a heartrate of 120+ while sitting:
    - Try and up the treshhold from its initial 750 (see 2nd picture), in the opposite case, you should decrease this value, I don't recommend anything below 512 though!

- If the script takes way too long to get a consistent value for your taste
    - Try lowering the last value in the 'rate' parameter (see 2nd picture),, it should give you a good result quicker, I don't recommend going below 4 here though.





```
def lees_pulse():
    print("Reading Pulse")
    # init variables
    rate = [0] * 10             # array to hold last 7 IBI values
    sampleCounter = 0           # used to determine pulse timing
    lastBeatTime = 0            # used to find IBI
    P = 512                     # used to find peak in pulse wave, seeded
    T = 512                     # used to find trough in pulse wave, seeded
    thresh = 750            # used to find instant moment of heart beat, seeded
    amp = 100                   # used to hold amplitude of pulse waveform, seeded
    firstBeat = True            # used to seed rate array so we startup with reasonable BPM
    secondBeat = False          # used to seed rate array so we startup with reasonable BPM
    old_BPM = 60
    BPM = 60
    global search_BPM
    search_BPM = 75

    IBI = 600                   # int that holds the time interval between beats! Must be seeded!
    # "True" when User's live heartbeat is detected. "False" when not a "live beat".
    Pulse = False
    lastTime = int(time.time()*1000)

    while True:

        Signal = int(MCP().read_channel(2))
        currentTime = int(time.time()*1000)

        sampleCounter += currentTime - lastTime
        lastTime = currentTime

        N = sampleCounter - lastBeatTime

        # find the peak and trough of the pulse wave
        # avoid dichrotic noise by waiting 3/5 of last IBI
        if Signal < thresh and N > (IBI/5.0)*3:
            if Signal < T:                          # T is the trough
                T = Signal                          # keep track of lowest point in pulse wave
```

1. Threshold
2. Rate

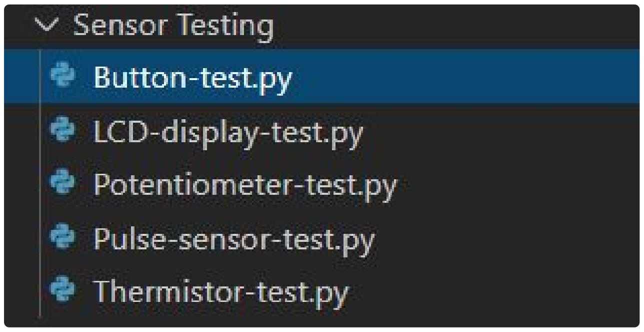
Step 18: Testing: Button

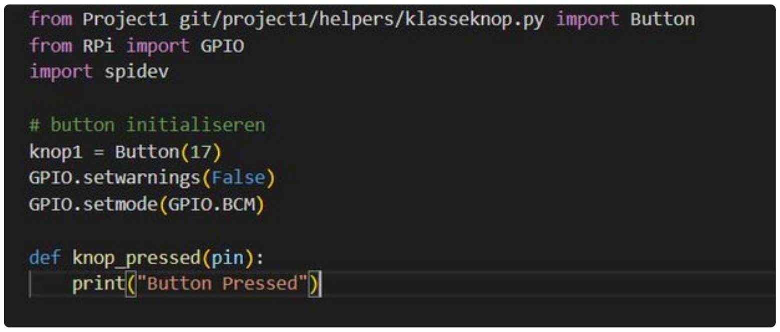
This is an easy one!

Open up the Button-test.py script

Run the script and press the button. If the press registers correctly you should see: "Button Pressed' pop up in the terminal.

If this doesn't work, the only thing I can recommend is checking your circuit and components. Maybe you have a broken wire? A broken resistor? maybe your button is not connected correctly?





```
from Project1 git/project1/helpers/klasseknop.py import Button
from RPi import GPIO
import spidev

# button initialiseren
knop1 = Button(17)
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

def knop_pressed(pin):
    print("Button Pressed")
```

Step 19: Testing: LCD Display

To test this component you should be extra careful, as it breaks easily:

First off check all your connections and wires once again, I can't stress this enough! One wrong connection can mess up your entire display!

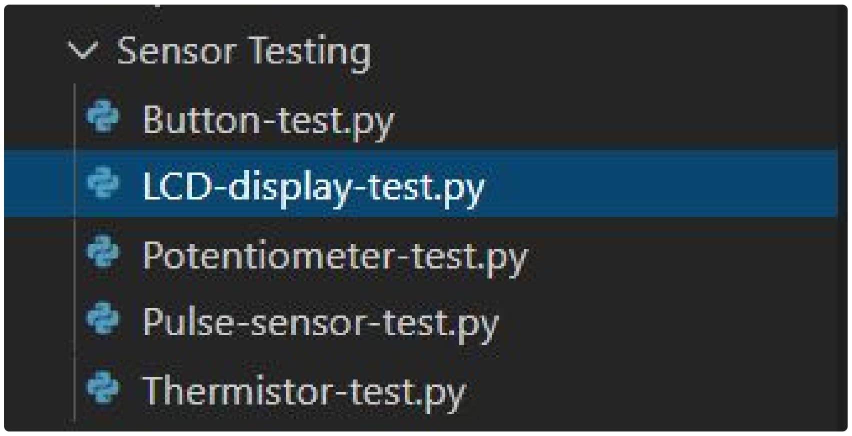
When you are absolutely sure everything is wired up correctly, power up your 5V breadboard power supply. The backlight of your LCD should already turn on.

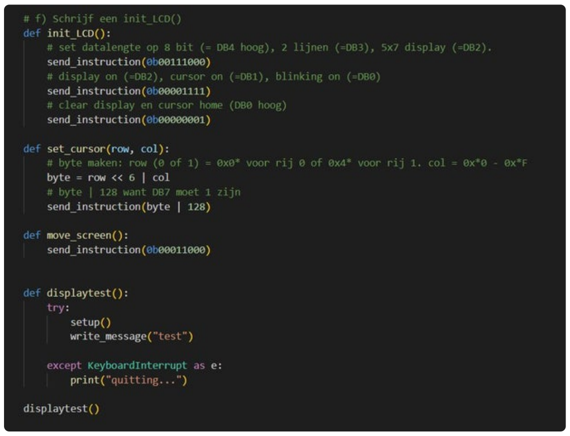
Now turn the knob/screw of the trimmer untill the upper row of your display is filled with little grey/white blocks, if this is the case the contrast of your display is configured correctly!

Now you can run the LCD-display-test.py

The white blocks should dissapear after a moment and the display should write the message "test" for you!

Feel free to change the message string in the displaytest method as this is a great way to pull pranks on your friends.





```
# f) Schrijf een init_LCD()
def init_LCD():
    # set datalengte op 8 bit (= DB4 hoog), 2 lijnen (=DB3), 5x7 display (=DB2).
    send_instruction(0b00111000)
    # display on (=DB2), cursor on (=DB1), blinking on (=DB0)
    send_instruction(0b00001111)
    # clear display en cursor home (DB0 hoog)
    send_instruction(0b00000001)

def set_cursor(row, col):
    # byte maken: row (0 of 1) = 0x0* voor rij 0 of 0x4* voor rij 1. col = 0x*0 - 0x*F
    byte = row << 6 | col
    # byte | 128 want DB7 moet 1 zijn
    send_instruction(byte | 128)

def move_screen():
    send_instruction(0b00011000)


def displaytest():
    try:
        setup()
        write_message("test")

    except KeyboardInterrupt as e:
        print("quitting...")

displaytest()
```

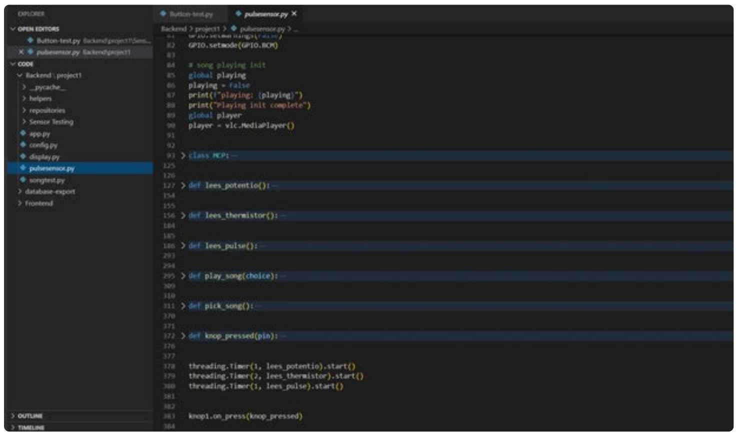
Step 20: HeartBeats: First Run

If all the seperate components seem to work, it's time to fire up the big boys!

Run the pulsesensor.py script, this is simply a combination (and extension) of all the seperate testing files.

Put your finger on the sensor and wait until you get a consistent heartrate. When that's the case, press the button!

you should see in the terminal that a song was picked from the embedded songlist and that it started 'playing', If you would like to listen to music right away? Just attach a speaker to your Pi via the Aux Port and you're good to go. Everything should work fine!



```python
GPIO.setmode(GPIO.BCM)

# song playing init
global playing
playing = False
print(f"playing: {playing}")
print("Playing init complete")
global player
player = vlc.MediaPlayer()


class MCP:...


def lees_potentio():...


def lees_thermistor():...


def lees_pulse():...


def play_song(choice):...


def pick_song():...


def knop_pressed(pin):...


threading.Timer(1, lees_potentio).start()
threading.Timer(2, lees_thermistor).start()
threading.Timer(1, lees_pulse).start()


knop1.on_press(knop_pressed)
```

Step 21: Front End

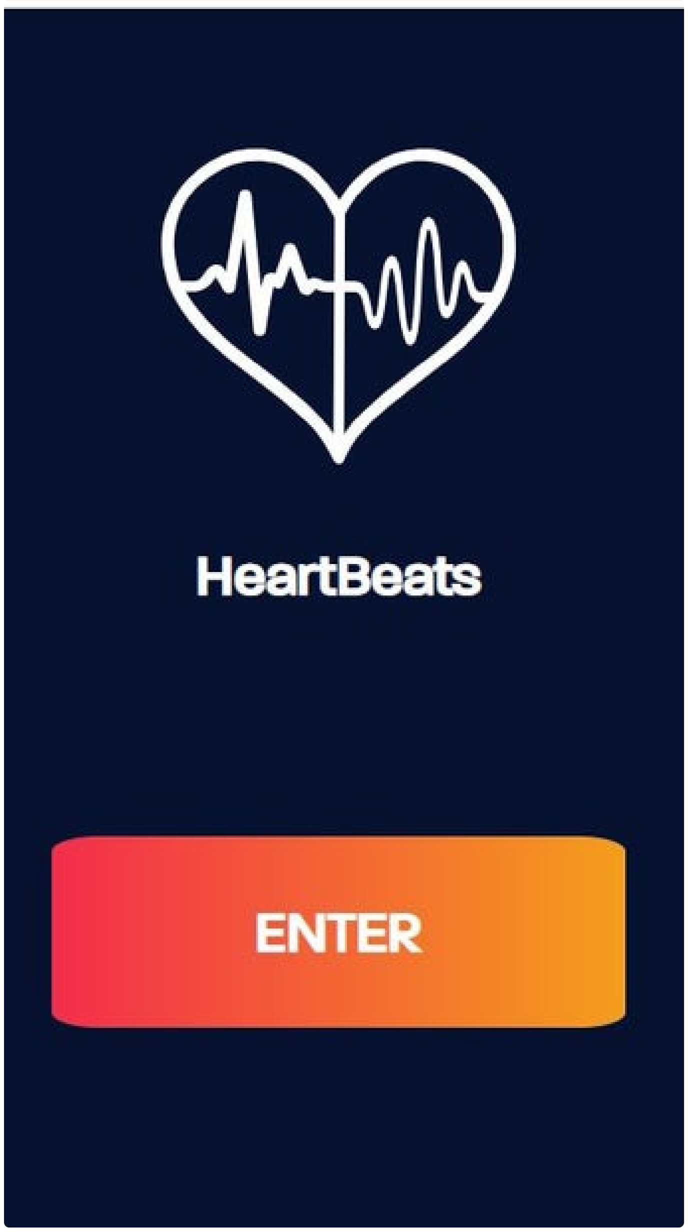
To make everything look/feel a little nicer, I also made a little website that you can use to control the site.

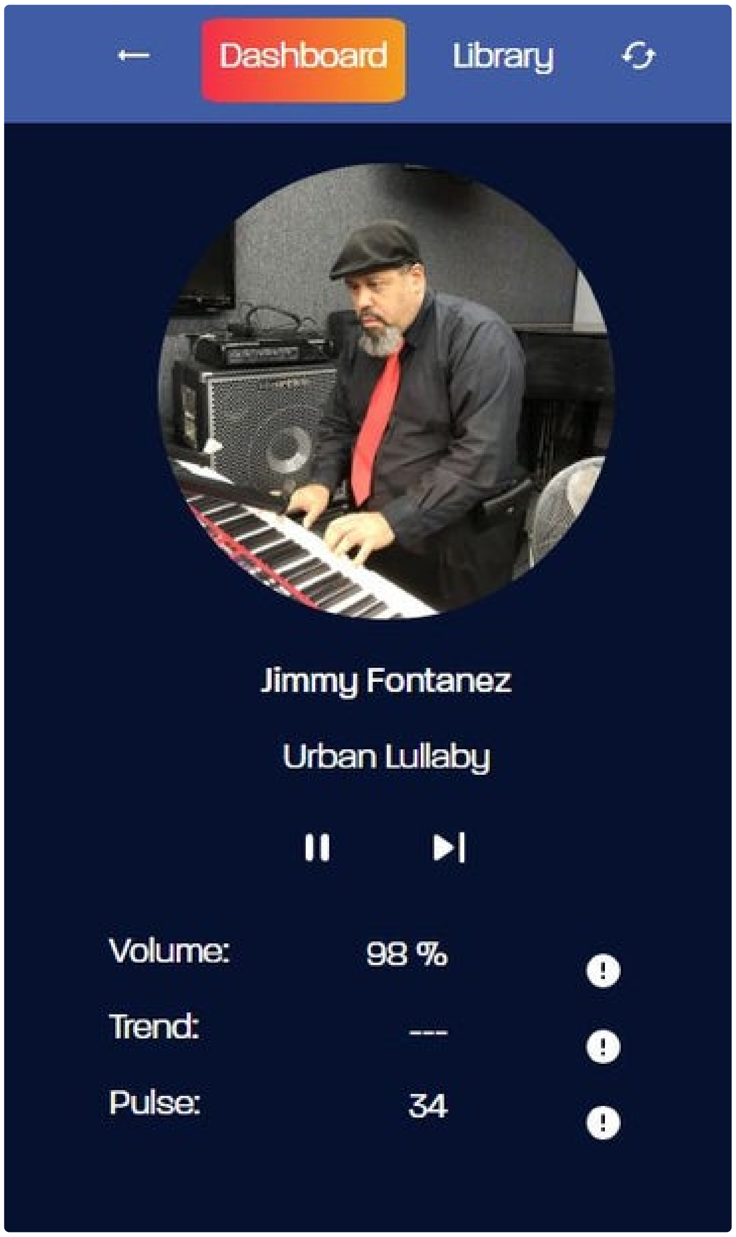
If you don't like the look of it! I encourage everybody to mess around with the HTML and CSS, this is another step where you can really make this project your own personal thing.

You can access the website by opening your favorite internet browser and looking up: 169.254.10.1

Make sure that you run the backend when using this site, otherwise it will remain empty and simply not function!

You can also access the library with all available songs on this website (and even delete the ones you don't like)

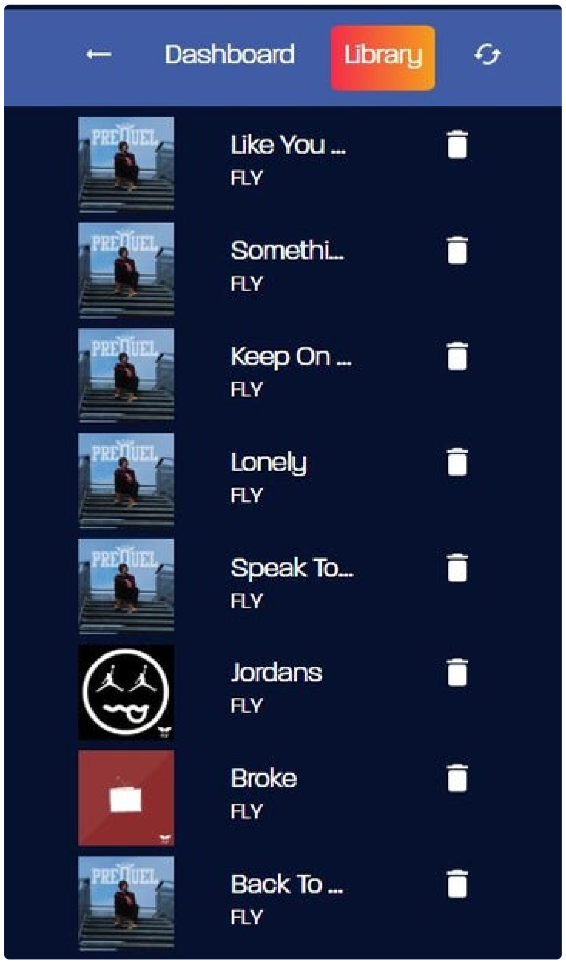




Jimmy Fontanez

Urban Lullaby

Volume:	98 %
Trend:	---
Pulse:	34



Like You ...
FLY

Somethi...
FLY

Keep On ...
FLY

Lonely
FLY

Speak To...
FLY

Jordans
FLY

Broke
FLY

Back To ...
FLY

Step 22: The Container

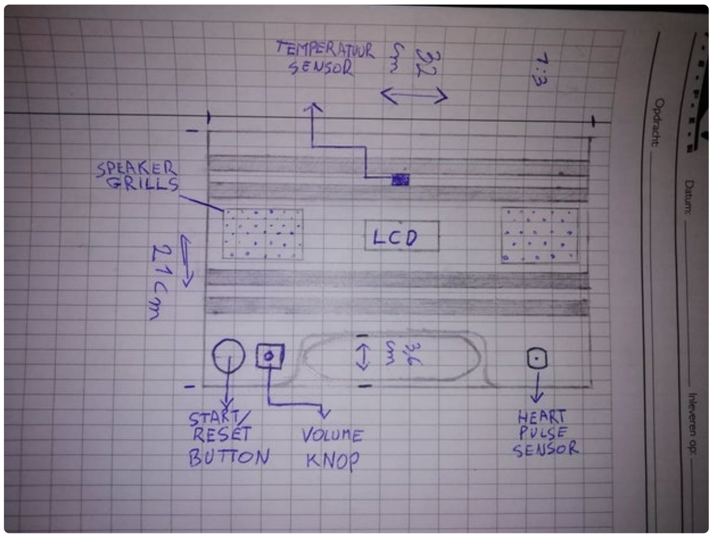
For this step, feel free to build all the parts into any container you'd like! The sky really is the limit for this part of the job

I do recommend securing all the pieces to a surface though (I achieved this with velcro) because the hearth pusle sensor is rather sensitive. The less everything is moving, the better!

I chose to build my project into an old storage box of one of my dads' drills. The plastic was robust enough to protect all my parts, but also soft enough to make it easy to reshape, cut pieces out, drill holes into,...

I made a quick 'n dirty sketch of how I wanted the final product to look, but this changed quite a bit when I actually started building. Here are some things to keep in mind:

- If you choose to build the speaker into the casing, that will be the only speaker you can use for this project without opening it up. (I have like 5 different speaker systems at home so this was a no go for me)
- Another downside to an integrated speaker is the added vibrations to the rest of the parts. This might mess with the accuracy of the Heart Pulse sensor
- I installed all my parts on 1 surface (the lid of the container) because otherwise I could easily break a cable just by opening up my project.

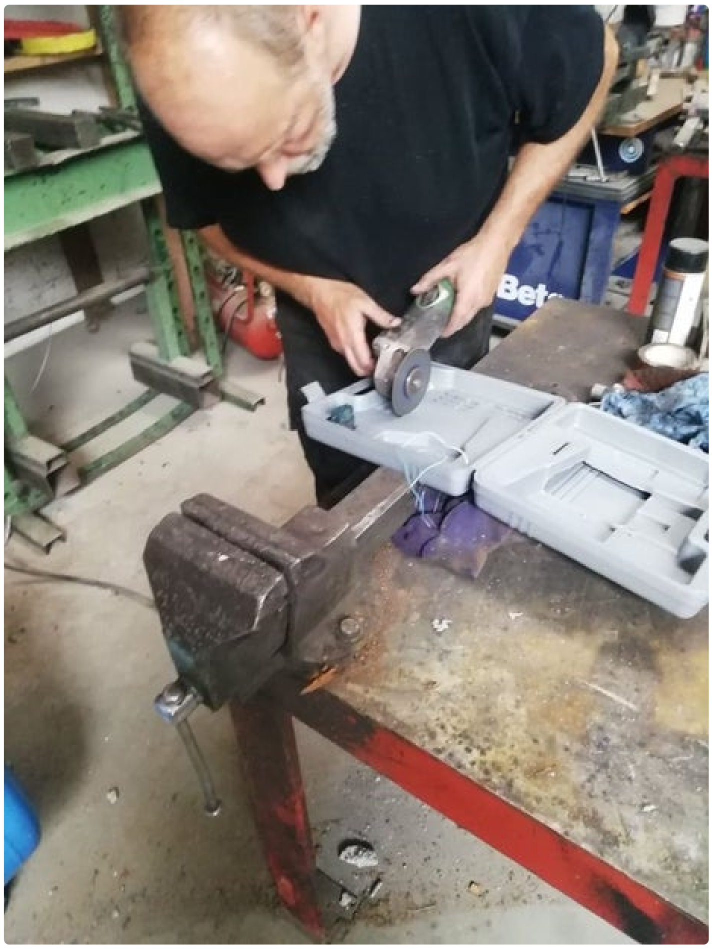


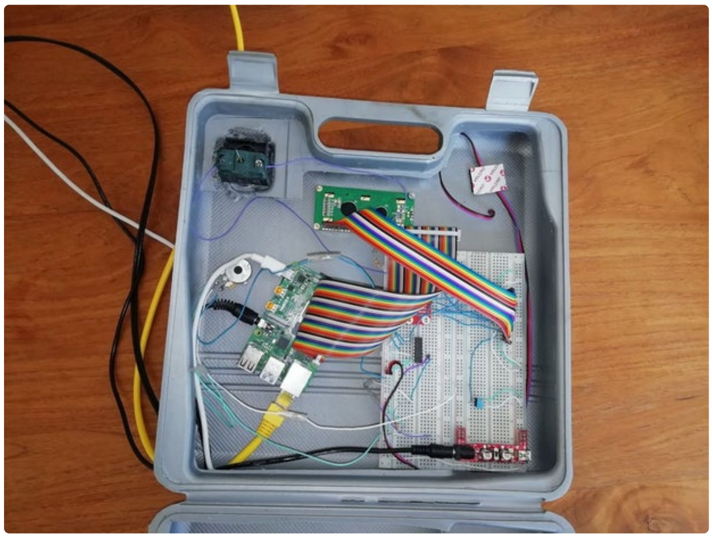
Step 23: Putting the Pieces Together

Time to get all our parts installed into our container! I mainly used a column drill to poke all the holes, and a disk grinder to grind out the hole for the LCD display.

Make sure to check all your connections again if you unplug/replug anything!

Now it's time to run some final tests and finish up some final details.





Step 24: Finishing Touches

This step is kind of personal for everybody, but what do you think you can still improve on the project?

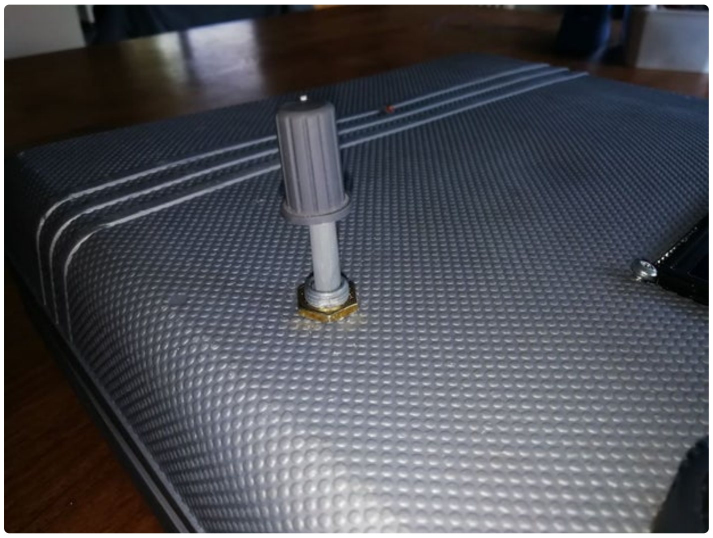
For me this was adding a real knob to the potentiometer, after shortening it's dial to size!

and attaching the screen a bit better and cleaner into the socket.

I hope you have fun and you learn a thing or two making this! Hit me up with all your versions of HeartBeats.

If you have any questions or comments: feel free to send me a message!

Bye for now



VSS VDD VO RS RW E D0 D1 D2 D3 D4 D5 D6 D7 A K