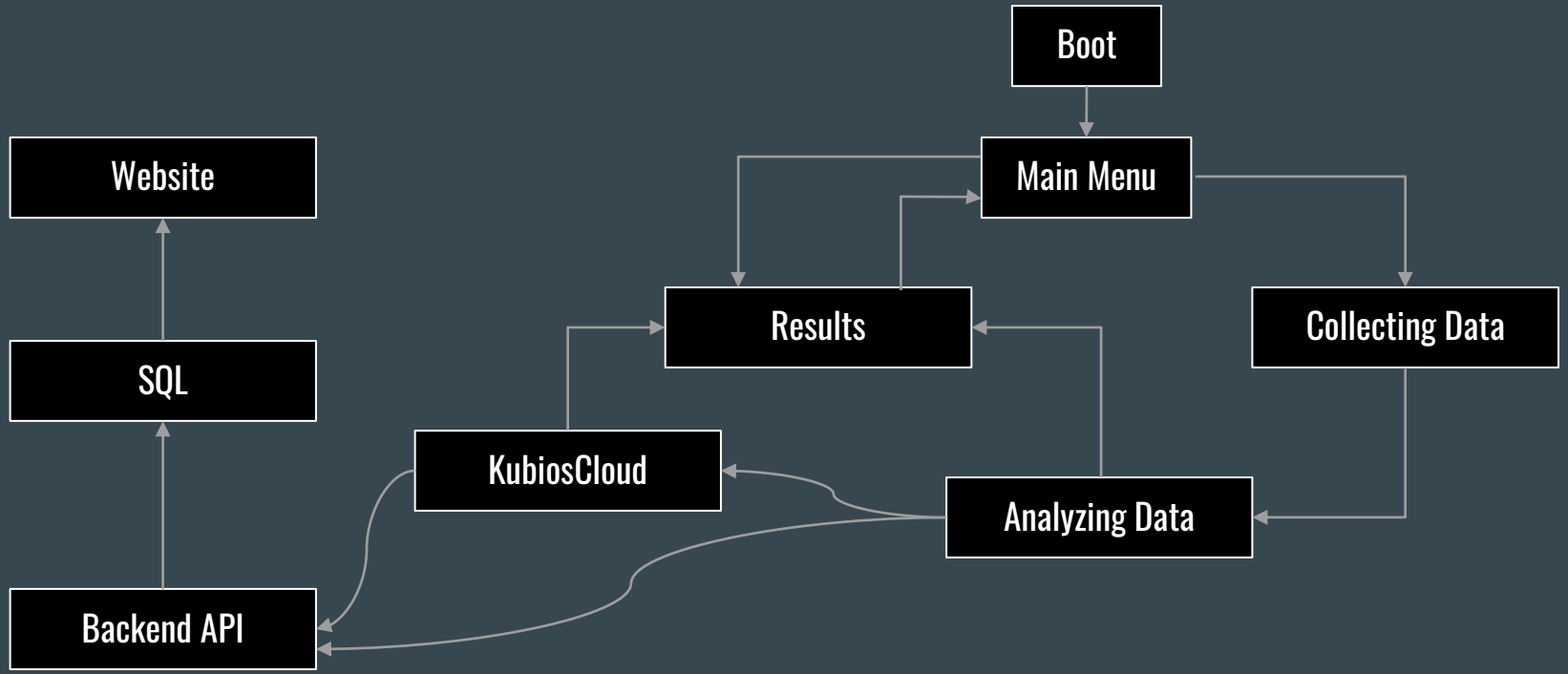
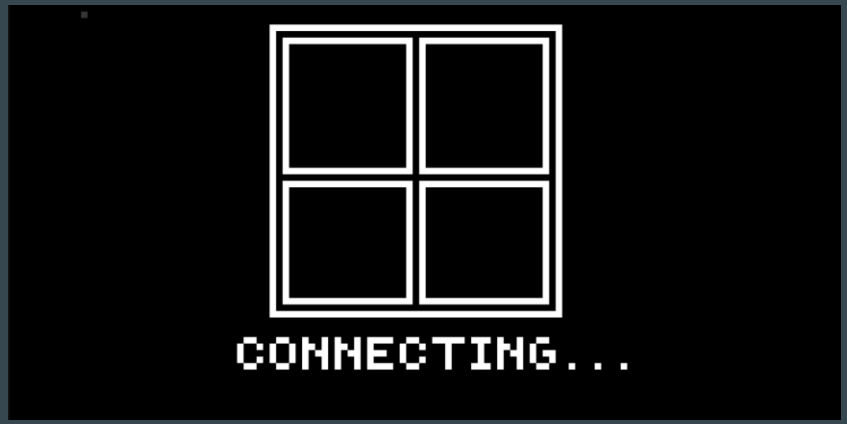
Heart Rate Monitor Project

Hardware 2


Boot
Website
Main Menu
Results
Collecting Data
SQL
KubiosCloud
Analyzing Data
Backend API



- Code that will run on boot
- Waits until WI-FI is connected

```
def boot_screen():
    global menu, enter

    boot_screen_design()
    enter = True
    menu = 1
```

```
import network
import secrets
import time

def connect_wifi():
    global wlan
    #Connect to WLAN
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.disconnect()
    wlan.connect(secrets.SSID, secrets.PASSWORD)
    ip = wlan.ifconfig()[0]

    return wlan
```

```
def boot_screen_design():
    global enter

    while enter == True:
        oled.display_rect(41,4,44,46)
        oled.display_rect(43,6,19,20)
        oled.display_rect(43,28,19,20)
        oled.display_rect(64,6,19,20)
        oled.display_rect(64,28,19,20)
        oled.show_screen()
        enter = False
    wLan = connect_wifi()
    while wlan.isconnected() == False:
        print('Waiting for connection...')
        oled.display_control(" Connecting.", 5, 55)
        time.sleep(1)
        oled.display_control(" Connecting..", 5, 55)
        time.sleep(1)
        oled.display_control(" Connecting...", 5, 55)
        time.sleep(1)
        oled.clear_control("            ..", 5, 55)
    ip = ask_wifi()
    enter = False
```

- Main Menu for the OS
- Handles starting data collection and going back to past results


```
def main_screen():
    main_screen_design()
```

```
def countdown(num):
    if enter == True:
        count = num
        for n in range(count):
            oled.display_control(f"{count}", 60, 20)
            time.sleep(1)
            oled.clear_control(f"{count}", 60, 20)
            count -= 1
```

```
def main_screen_design():
    global enter

    if enter == True:
        oled.clear()
        oled.display_fill(1, 49, 42, 15)
        oled.display_fill(90, 49, 45, 15)
        oled.clear_control("Start", 2, 53)
        oled.clear_control("Back", 94, 53)
        enter = False
```

```
while True:
    while menu == 0:
        boot_screen()

    while menu == 1:
        main_screen()
        button_listener()

    while menu == 2:
        collecting_screen()
        button_listener()

    while menu == 3:
        analyzing_screen()
        button_listener()

    while menu == 4:
        result_screen()
        button_listener()

    while menu == 5:
        result_screen_2()
        button_listener()
```

```
def button_listener():
    global menu, enter, sw0_past_state, sw1_past_state

    if sw0_past_state != main_3_3v.value() or sw1_past_state != alt_3_3v.value():
        if menu == 1 and main_3_3v.value() == 1: #from main menu to collect data
            sw0_past_state = main_3_3v.value()
            enter = True
            menu = 2
            while main_3_3v.value() == 1:
                pass
        if menu == 1 and alt_3_3v.value() == 1: #from main menu to past result
            sw1_past_state = alt_3_3v.value()
            enter = True
            menu = 4
            while alt_3_3v.value() == 1:
                pass
```

- Collects values from the sensor
- Detects peaks from the data
- Calculates PPI values
- Runs for 30 to 90 seconds


```
def collecting_data_design():
    global enter

    if enter == True:
        oled.clear()
        oled.display_control("  Collecting...", 5, 30)
        oled.display_fill(74, 49, 48, 15)
        oled.clear_control("Cancel", 75, 53)
        enter = False
```

```
def collecting_screen():
    global menu, enter, results, oled, ppi_list
    countdown(5)
    collecting_data_design()
    received_data = collect_data(30)
    results = received_data[0]
    ppi_list = received_data[1]
    print(ppi_list)
    enter = True
    menu = 3
```

```
def bpm_calc():
    global bpm, bpm_int, ppi_list, bpm_list, bpm_avg

    if len(l) > 2:
        ppi = l[-1] - l[-2]
        if ppi > 500 and ppi < 1500:
            if ppi != ppi_list[-1]:
                ppi_list.append(ppi)
                bpm = 60/(ppi/1000) # equation to determine bpm
            bpm_int = int(bpm)
            bpm_list.append(bpm_int)
            bpm_avg = round(sum(bpm_list)/len(bpm_list))
            bpm_calc_screen(bpm_int)
    #print(l)
    return bpm_avg, ppi_list

def bpm_calc_screen(value):
    global prev_bpm, enter
    if prev_bpm != value:
        oled.clear_control(f"{prev_bpm}", 45, 15)
        oled.display_control(f"{value}  BPM", 45, 15)
    if enter == True:
        oled.display_control("  Collecting...", 5, 30)
        oled.display_fill(74, 49, 48, 15)
        oled.clear_control("Cancel", 75, 53)
        enter = False
    prev_bpm = value
    return
```

```
def collect_data(run_time):
    global samples, avg, avg_samples, l, beats, bpm_list
    l = []
    bpm_list = []
    start_time = time.time()
    while True:
        current_time = time.time()
        if not samples.empty():
            value = samples.get()
            avg_samples.put(value)
            avg = sum(avg_samples.data)/ avg_w
            avg_samples.get()
            thres_min = min(samples.data)
            thres_max = max(samples.data)
            threshold = thres_max -(thres_max - thres_min ) * 0.25

        if avg > threshold and beats == False:
            beats = True
            l.append(time.ticks_ms())

        if avg < threshold and beats == True:
            beats = False

        result = bpm_calc()

        if current_time - start_time >= run_time:
            l = []
            return result
```

- Turn PPI information into BPM, SDNN and RMSSD values
- Sends the PPI list to the KubiosCloud to receive the SNS and PNS indexes

```
def analyzing_screen():
    global menu, enter, results, ppi_list, analyzed_local, analyzed_kubios

    if enter == True:
        print(results)

    analyzing_data_design()

    if len(ppi_list) > 10:
        analyzed_kubios = kubios_call(ppi_list)
    analyzed_local = kubios_backup(ppi_list)

    enter = True
    menu = 4
```

```
def kubios_backup(data):
    #bpm
    hr_ppi = sum(data)/len(data)
    hr_bpm = round(60 / hr_ppi * 1000)
    #sdnn
    squared_diffs = [(x - hr_ppi)**2 for x in data]
    sum_squared_diffs = sum(squared_diffs)
    variance = sum_squared_diffs / (len(data) - 1)
    sdnn = round(variance**0.5)
    #rmssd
    squared_diffs = [(data[i] - data[i-1])**2 for i in range(1, len(data))]
    mean_squared_diffs = sum(squared_diffs) / len(squared_diffs)
    rmssd = round(mean_squared_diffs ** 0.5)

    return hr_bpm, sdnn, rmssd
```

```
def analyzing_data_design():
    global enter

    if enter == True:
        oled.clear()
        oled.display_control("  Analyzing...", 5, 30)
        oled.display_fill(74, 49, 48, 15)
        oled.clear_control("Cancel", 75, 53)
        enter = False
```

```
def kubios_call(intervals):
    APIKEY = "pbZRUi49X48I56oL1Lq8y8NDjq6rPfzX3AQeNo3a"
    CLIENT_ID = "3pjgjdmamlj759te85icf0lucv"
    CLIENT_SECRET = "111fqsli1eo7mejcrlffbklvftcnfl4keoadrdv1o45vt9pndlef"
    LOGIN_URL = "https://kubioscloud.auth.eu-west-1.amazoncognito.com/login"
    TOKEN_URL = "https://kubioscloud.auth.eu-west-1.amazoncognito.com/oauth2/token"
    REDIRECT_URI = "https://analysis.kubioscloud.com/v1/portal/login"
    response = requests.post(
        url = TOKEN_URL,
        data = 'grant_type=client_credentials&client_id={}'.format(CLIENT_ID),
        headers = {'Content-Type':'application/x-www-form-urlencoded'},
        auth = (CLIENT_ID, CLIENT_SECRET))
    response = response.json()
    access_token = response["access_token"]
    #intervals = [828, 836, 852, 760, 800, 796, 856, 824, 808, 776, 724]


    data_set = {
       "type": "PPI",
       "data": intervals,
       "analysis": {
         "type": "readiness"
                 }}

    response = requests.post(
                            url = "https://analysis.kubioscloud.com/v2/analytics/analyze",
                            headers = { "Authorization": "Bearer {}".format(access_token),

                                        "X-Api-Key": APIKEY },
                            json = data_set)
    json_list = response.json()
    parsed_values = [json_list["analysis"]["sns_index"], json_list["analysis"]["pns_index"]]
    return parsed_values
```

- Checks if new or previous data can be found

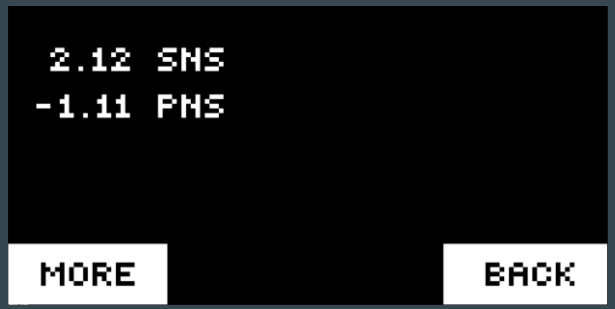
```
def result_screen():
    global results, analyzed_local, analyzed_kubios

    if results != None:
        result_screen_design(results, analyzed_local[1], analyzed_local[2])
    else:
        result_screen_design("--", "--", "--")

def result_screen_2():
    global results, analyzed_local, analyzed_kubios

    if analyzed_kubios != None:
        sns = f"{(analyzed_kubios[0]):.2f}"
        pns = f"{(analyzed_kubios[1]):.2f}"
        result_screen_2_design(sns, pns)
    else:
        result_screen_2_design("--", "--")
```

- Presents the values received from the local calculations

```
def result_screen_design(bpm, sdnn, rmssd):
    global enter

    if enter == True:
        oled.clear()
        oled.display_control(f"{bpm}     BPM", 5, 5)
        oled.display_control(f"{sdnn}    SDNN", 5, 17)
        oled.display_control(f"{rmssd}   RMSSD", 5, 30)
        oled.display_fill(1, 49, 42, 15)
        oled.clear_control("More", 2, 53)
        oled.display_fill(90, 49, 45, 15)
        oled.clear_control("Back", 94, 53)
        enter = False
```

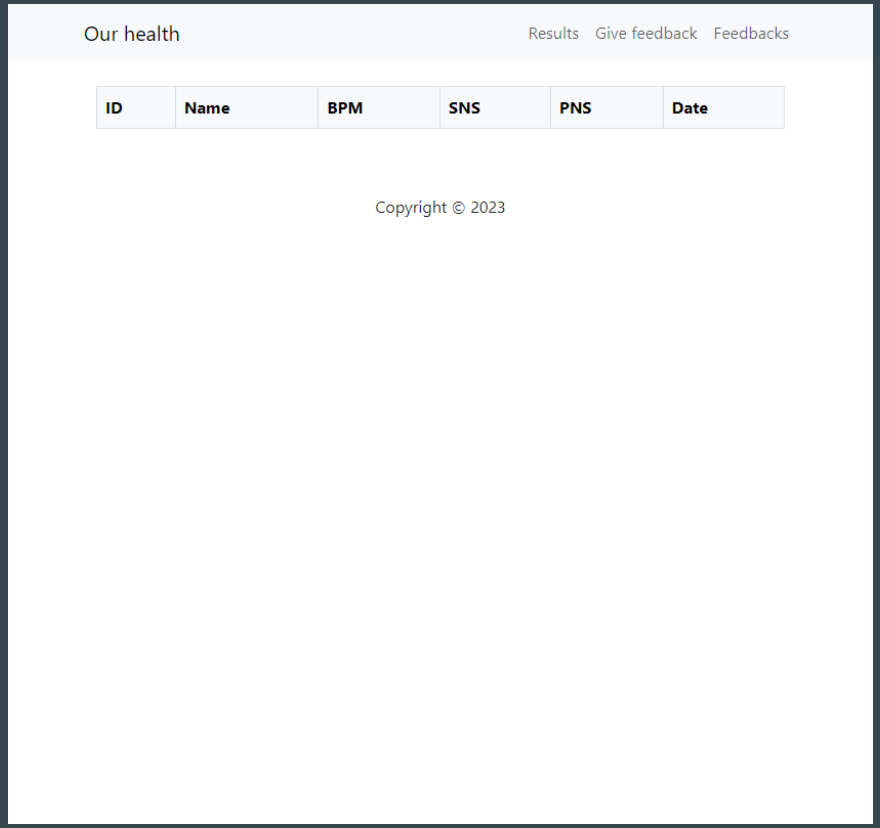
- Presents the values provided by KubiosCloud

```
def result_screen_2_design(sns, pns):
    global enter

    if enter == True:
        oled.clear()
        oled.display_control(f"{sns}   Stress", 5, 5)
        oled.display_control(f"{pns}  Recovery", 5, 17)
        oled.display_fill(1, 49, 42, 15)
        oled.clear_control("More", 2, 53)
        oled.display_fill(90, 49, 45, 15)
        oled.clear_control("Back", 94, 53)
        enter = False
```

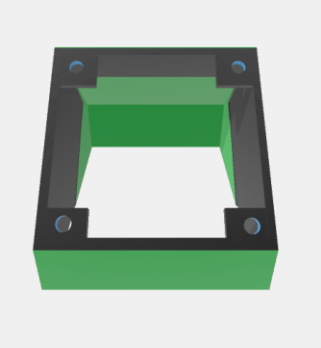


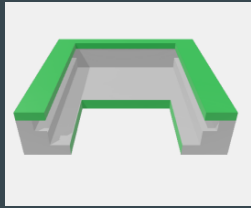
Premade functions to control the screen

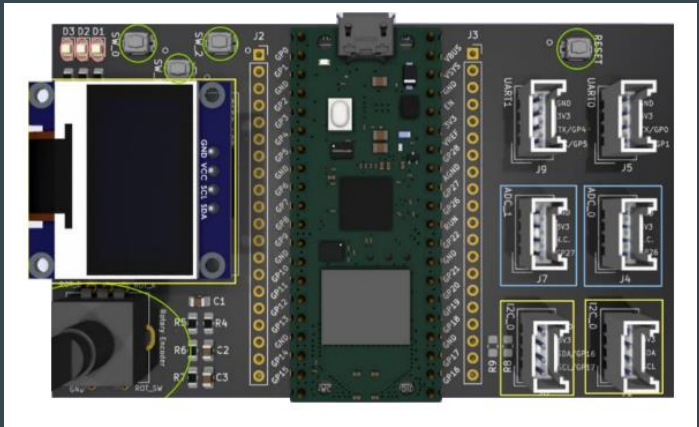
```
from machine import Pin, I2C
import ssd1306


i2c = I2C(1, sda=Pin(14), scl=Pin(15),freq=400000)
display = ssd1306.SSD1306_I2C(128, 64, i2c)

class Oled:
    def __init__(self, id):
        self.name = id

    def display_control(self, text, x, y):
        display.text(text, x, y)
        display.show()

    def display_fill(self, start_x, start_y, end_x, end_y):
        display.fill_rect(start_x, start_y, end_x, end_y, 1)
        display.show()

    def clear_control(self, text, x, y):
        display.text(text, x, y, 0)
        display.show()

    def clear(self):
        display.fill(0)
        display.show()

    def pixel(self, x, y, c):
        display.pixel(x, y, c)
        display.show()

    def display_rect(self, start_x, start_y, end_x, end_y):
        display.rect(start_x, start_y, end_x, end_y, 1)

    def show_screen(self):
        display.show()
```



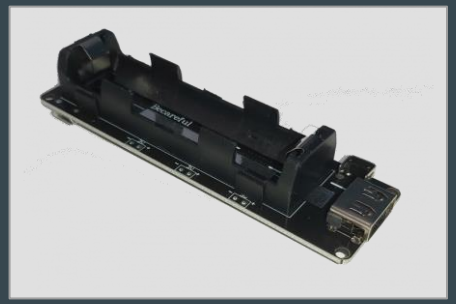
- Prototype Website up locally
- Patient data is retrieved from SQL Database

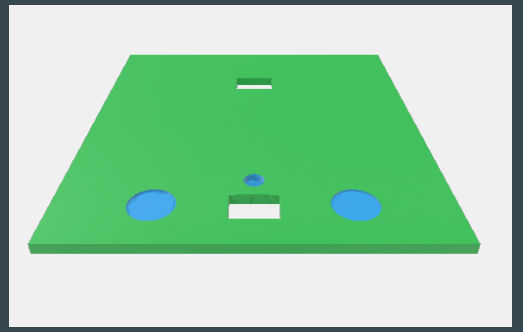




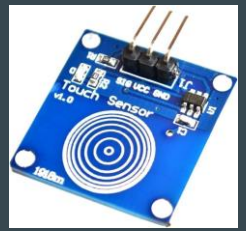


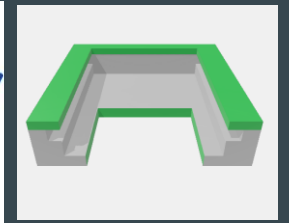
- SSD1306 Oled screen
- CrowTail Pulse Sensor v2.0
- Raspberry Pi Pico microcontroller
- Custom connector board by J. Hotchkiss






```
main_3_3v = Pin(1, Pin.IN)
alt_3_3v = Pin(0, Pin.IN)
```





give us Feedback

view —--->