IOT pulse measuring system

Project :   IOT pulse measuring system
Deltager :   Tobias Valbjørn
Projekt startdato: 27-aug-2018
Projekt slutdato: 14-dec-2018

Introduction

This report covers the 5th semester IOT project at EDE Herning in the fall by Tobias Valbjørn. It describes the development of a prototype of an IOT pulse measuring device, that can give you a general idea of your fitness level based on your resting heart rate. The project scope and requirements are described in Aarhus University course catalog for the course E5IOT.

Problem statement

The resting heart rate is the speed of the heart beat, when you are rested.

"As you get fitter, your resting heart rate should decrease. This is due to the heart getting more efficient at pumping blood around the body, so at rest more blood can be pumped around with each beat, therefore less beats per minute are needed. "[1]

It could be interesting to track the resting heart rate over a period of time, to get a general indication of fitness.

"The measurement of resting heart rate or pulse rate (the number of heart beats per minute) should be taken after a few minutes upon waking whilst still lying in bed. Give your body some time to adjust to the change from sleeping before taking your pulse (2-5 minutes). "

The measurement of the pulse first thing in the morning can be a challenge if you don't have a wrist heart rate monitor, which can be expensive. If you use a chest strap, your resting heart rate will increase as you put it on. You can also use your fingers and a stopwatch, which can be difficult and inaccurate. In most of the cases you will have to remember the resting heart rate and note it manually.

Project description

The project focuses on improving and automating the process of keeping track of the resting heart rate.

The idea is that upon waking up, you would turn on your simple IOT resting heart rate monitoring system consisting of a particle photon, simple arduino heart rate sensor, and a status LED. The whole system is connected to your home Wifi.

Within a minute, as your body adjusts to change from sleeping, the system will be ready. Using the heart rate monitor your resting heart rate would be monitored, and uploaded to Thingsspeak. You would get visual feedback from the LED, when the test is done, and the

---

[1] https://www.topendsports.com/testing/heart-rate-resting.htm

results are uploaded to the web. When taking the test, the LED will blink in a fading heart rate pattern.

You could afterward see a graph of your average heart rate over time on Thingsspeak. You can see if it has been increasing or decreasing, and you could use it to estimate your fitness level into a basic category according to your sex and age, as seen on figure 1.

Resting Heart Rate for MEN

	Age 18-25	26-35	36-45	46-55	56-65	65+
Athlete	49-55	49-54	50-56	50-57	51-56	50-55
Excellent	56-61	55-61	57-62	58-63	57-61	56-61
Good	62-65	62-65	63-66	64-67	62-67	62-65
Above Average	66-69	66-70	67-70	68-71	68-71	66-69
Average	70-73	71-74	71-75	72-76	72-75	70-73
Below Average	74-81	75-81	76-82	77-83	76-81	74-79
Poor	82+	82+	83+	84+	82+	80+

Figure 1. General classification of fitness based on resting heart rate.

Further work

The data from thingsspeak could be used in a web interface. Where the user could insert user information, and the web interface could for example use the information to automatically place the user in a basic category and show progress. To get a general idea, see figure 2.
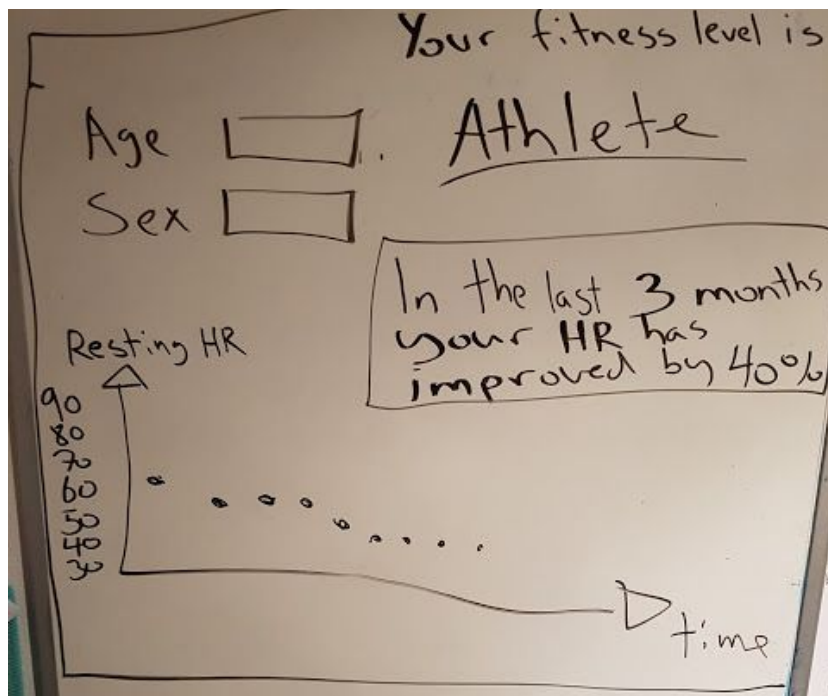

Figure 2. Demo webinterface.

Design requirements

The system will be turned on during the measurements. When the results have been read, the system will be turned off immediately. In this project scope to make a simple prototype and keep the budget low, USB power from the wall socket can be used instead of a battery powered system. Most people have a socket close to the bed for charging the phone or a night lamp for reading.
It does not need to take any sleep-mode functionality into account.
It only needs to start up fast.
It needs to be small to be in the bedroom, near the bed, supposedly at a bedside table.
The hardware needs to be organised in a way, that it can be easily equipped upon waking up.

Functional requirements

Need to have:

1. The platform has Wifi connectivity, and the system will connect over Wifi.
2. When the Wifi has been set up, the system should automatically connect to the internet when power is turned on.
3. After power is on, the application should start running automatically.
4. The platform has available digital or analog I/O to connect sensors and actuators.
5. The system reads the resting heart rate from a heart rate sensor local to the device.
6. The system should be able to send the heart rate locally from the particle photon to thingsspeak.
7. The device controls an actuator in the form of an LED.
8. The device uses data from thingsspeak to augment what it does, by giving the user feedback on the LED.
9. The webservice should be able to show the results of the heart rate test on thingsspeak.

Nice to have:

10. The system should be able to measure the heart rate accurately. Within 2 bps difference from a Garmin Vivosport heart rate monitor.
11. The whole system should be able to fit into a box with the maximum dimensions of length 15 cm, width 10 cm, height 8 cm.
12. The system should be able to store the necessary information of 1 user from the website. (age, sex, previous heart rate measurements, category, and date)
13. The webservice should be able to compare the heart rate data with the heart rate table.
14. The webservice should be able to compare the heart rate data with previous data.

Non-functional requirements

- The system needs to be simple, with as few user interactions as possible.
- The system functionality will be focused on the task of measuring the resting heart rate.
- Existing standards and protocols will be used wherever possible.
- The system will be calm when everything is working fine. It will not bring attention to itself beside when giving the user instruction or feedback.

Analysis and design

Development platform

Particle photon

Pros	Cons
Small form factor	Limited amount of GPIO
Wi-fi	Limited amount of memory
Well documented	Limited power output
Prior experience with platform	Bound to particle platform
Education is based on platform	
Availability of sparring about platform from teachers.	

The particle photon can be used, since the project has simple needs, and a small form factor would be prefered. There needs to be Wi-fi and a few GPIO's. We don't need a lot of

GPIO's, memory, high processing speed, or other communication modules. Furthermore the device does not have to be able to deliver power to a lot of sensors and actuators, only one heart rate sensor, and a LED, where the particle photon will be sufficient. The particle photons tasks will be to measure an analog value from the heart rate, publish the analog value, get basic response from the internet, and control the LED.
I can use the particle photon from the class set.

Interface analysis

If I use the Vin I can supply 1 A, and if I use 3V3 I can use 100 mA.[2] I can maximally draw 25 mA from one pin, and from all the pins it can maximally draw/deliver 120 mA.

General requirements for sensors and actuators

Any sensor or actuator cannot draw more than 25 mA from one I/O pin.
In total the current Input/Output from the pins cannot exceed 120 mA.

Sensors

I need my system to interface with a sensor. The information we need from the outside world are a resting heart rate from a person converted to an electric signal. The particle photon has 8 A/D inputs (A0-A7), and we can choose anyone of these.

General information about sensors

There are mainly two kind of heart rate sensors that I will focus on. It is chest strap and optical sensors.

Chest strap
Chest strap heart-rate monitors use electrocardiography to record the electrical activity of the heart. A pad with electrodes records the electrical activity of the heart, but the pad needs to be moist for proper conduction of the electrical signal. It is both expensive and impractical since the system needs to be used when waking up. Applying moist on the sensor is not an option.

Optical sensors
Another technology is using an optical sensor, and this is the most common in wearable heart rate monitors. [3] They use photoplethysmography (PPG). It is the use of light to capture the flow of blood through the veins. Often they use LEDs with green light unto the skin. The different wavelength of the light will interact with the flowing blood, and the reflection of the light will be registered in a sensor. I is an integrated sensor, the received signal are amplified

---

[2] https://docs.particle.io/datasheets/photon-(wifi)/photon-datasheet/
[3]
https://arstechnica.com/gadgets/2017/04/how-wearable-heart-rate-monitors-work-and-which-is-best-for-you/

and noise cancellation is applied. The sensor output is an analog electronic signal. The optical sensors can be less reliable since they are placed further from the heart. It is an acceptable trade off, but the accuracy will be tested. Since there is amplification involved we will need an active sensor, which requires an external power supply.

Design requirements for heart rate sensor

- It needs to be inexpensive, so it can be used in a low budget prototype system.
- It needs to be compatible with the particle photon, and within the general requirements for sensors and actuators.
- It needs to be well tested and well documented.
- I am looking for one to put on a finger or wrist.
- I need an optical heart rate sensor
- It has to include amplification and noise cancelation
- It has to be easy to apply.

Selection criterion

1. Accuracy
   - Difference between the measured value and the "true" value.
   - It has to be within 2 difference from a wrist heart rate monitor.

2. Precision
   - The ability to reproduce with a given accuracy
   - Precision can be difficult to measure with resting heart rate, since subjective factors can change the outcome of the measurement. The heart rate could increase or decrease while measuring. Despite that the sensor needs to have precision. A counter measure could be to measure over time and observe the beats per minute over several minutes.
   - If we measure the heart rate over 5 minutes the bpm for each minute should not deviate more than 3 bpm.
3. Resolution
   - The smallest change the sensor can differentiate.
   - The sensor needs to be able to measure a difference of 1 bpm.
4. Response time
   - The time lag between the input and output
   - The sensor needs to receive the analog signal from the sensor immediately.
   - I am working with electric signals, and I assume that this will not be a problem. The only test I will make of the response time is, if there is a visible lag of more than 1 second.

SEN-11574

There is a well documented plug and play heart rate sensor for arduino, SEN-11574[4], which means it is compatible with the particle photon. It has a lot of code to get started. It includes velcro, ear plugs, and other stuff to help with the measurements as seen on figure 3:

Figure 3 SEN-11574 pulse sensor kit.

The maximum ratings has to be compatible:

It can be used with 3.3 V and 5V, and it has a supply current of 3-4 mA which is well within the boundaries of our system.

It has 3 pins. It can be connected to the particle photon in the following way:

Sensor	Particle photon
Red wire	5V
Black wire	GND
Purple wire	A0

I use the 5V as it can deliver most current to the connected devices.

The price is: $24.95 = 161 dkk, which meets the requirement of low budget.

For description of the sensor, you can get it on their website.[5]

---

[4] https://www.sparkfun.com/products/11574

[5] https://pulsesensor.com/pages/open-hardware

Documentation of heart rate sensor

I have been looking at the documentation for the sensor, and from a quick search it seems that the sensor is well used and described in hobbyist projects:

The sensor is made by "World Famous Electronics", which actively maintains example projects and code at:
https://pulsesensor.com/

Getting started code:
https://github.com/WorldFamousElectronics/PulseSensorStarterProject

Getting advanced code:
https://github.com/WorldFamousElectronics/PulseSensor_Amped_Arduino

Pulse sensor processing visualizer:
https://github.com/WorldFamousElectronics/PulseSensor_Amped_Processing_Visualizer

Project with code and thingsspeak integration:
https://circuitdigest.com/microcontroller-projects/iot-heartbeat-monitoring-using-arduino

Conclusion

The SEN-11574 meets the design requirements and are chosen for the project. Given the information I have about the sensor, I can only guess that the chosen sensor meets the selection criteria, it must be tested during the development of the prototype.

Actuators

For the system to be in accordance with the project requirements, there needs to be actuators that acts on information from a web service. To keep it simple, the actuator will show the status of the process. I need an actuator that can inform the user of the status of the project. When is the system measuring? When is the system done, and have received acknowledgement from the web server? An LED can do the job. It will not disturb the user, and it is lightweight. I considered using the onboard LED, but it was not sufficient to display all the different information.

Red LED

The LED used for the project is a basic red LED from the lab, and it operates on 10 mA. It is well within the limits for the particle photon. I can use any one of the PWM pins. I have chosen a red LED, so that it resembles a beating heart. It blinks in a fading heart rate pattern.

Several types of actuators have also been considered. A LCD-display, RGB LED, and a vibrator motor.

LCD display

Pros	Cons
Opportunity for graceful degradation	Bigger and more complex system
Clear instructions	Greater requirement for power.
	Greater requirement for pins
	Higher price.
	May add unwanted information.

The system's main purpose is to measure the heart rate, the screen is evaluated to be redundant, seeing that I want to show the information more visually pleasing on a website anyway. If the user wants information afterwards, he can just visit the website on his smartphone.

RGB LED

The RGB LED could show the status throughout the whole process in greater detail than an ordinary LED with the use of different colors.

Pros	Cons
Colors to display status clearly	More components
	Added complexity
	Takes up more space
	May include redundant information

The pro of displaying the status more clearly was disputed through user testing, where the simple red LED did the job of displaying the status of the system, along with the status LED from the particle photon, did the job sufficiently.

Vibrator motor

With his eyes closed the user could get feedback from a haptic sensor throughout the pulse measurement, as well as a signal when the measurement is done. A vibrator motor can be used to provide the haptic feedback.

Pros	Cons
Take test with eyes closed	Vibration could disturb the test.
	High operating current of 70 mA and it uses 3.3V terminal
	Will use around 70% of total current.
	More components
	Added complexity
	Takes up more space
	May include redundant information

It is evaluated that the ability to take the test with the eyes closed, with the possibility of getting a lower resting heart rate is not worth the down sides.

Overview

A complete table of the particle photon pins and the devices that are connected to them:

Particle Photon Pins	Device using
VIN	Heart rate sensor, red LED
GND	All devices
A2	Heart rate sensor
D3	Red LED

A simple overview of the logic in the system:

Quantity being measured	Input Devices (sensors)	Output devices (Actuators)
Heart rate	Heart rate sensor	red LED

Choice of webservice

The design requirements for the webservice is:
- The ability to work with the particle photon
- Display uploaded data visually on a graph
- The ability to extract the information from thingsspeak to a different platform

Thingsspeak meets these requirements, and we have experience with the platform from IOT.

11/27

Diagrams

Figure 4 shows the architecture of the system. Figure 5 shows a simple use case diagram, with the flow through the system. A webhook might be used as middleman between the local system and the webservice, thingsspeak.
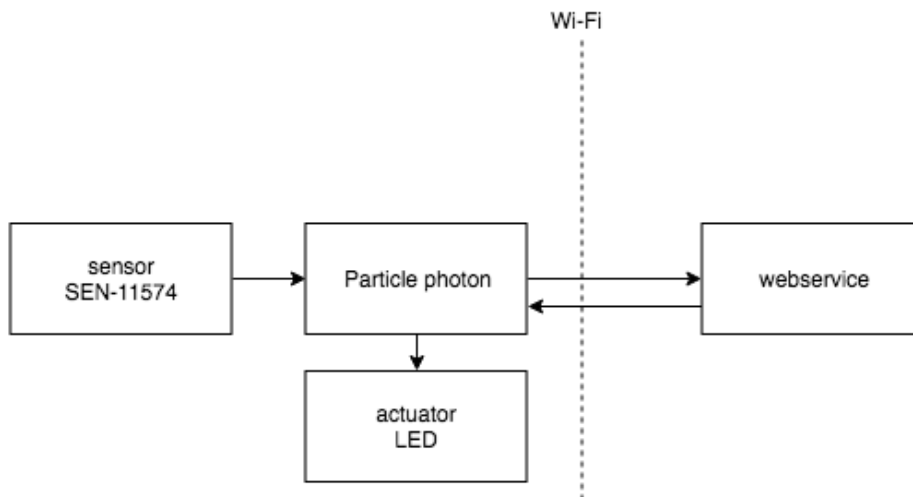
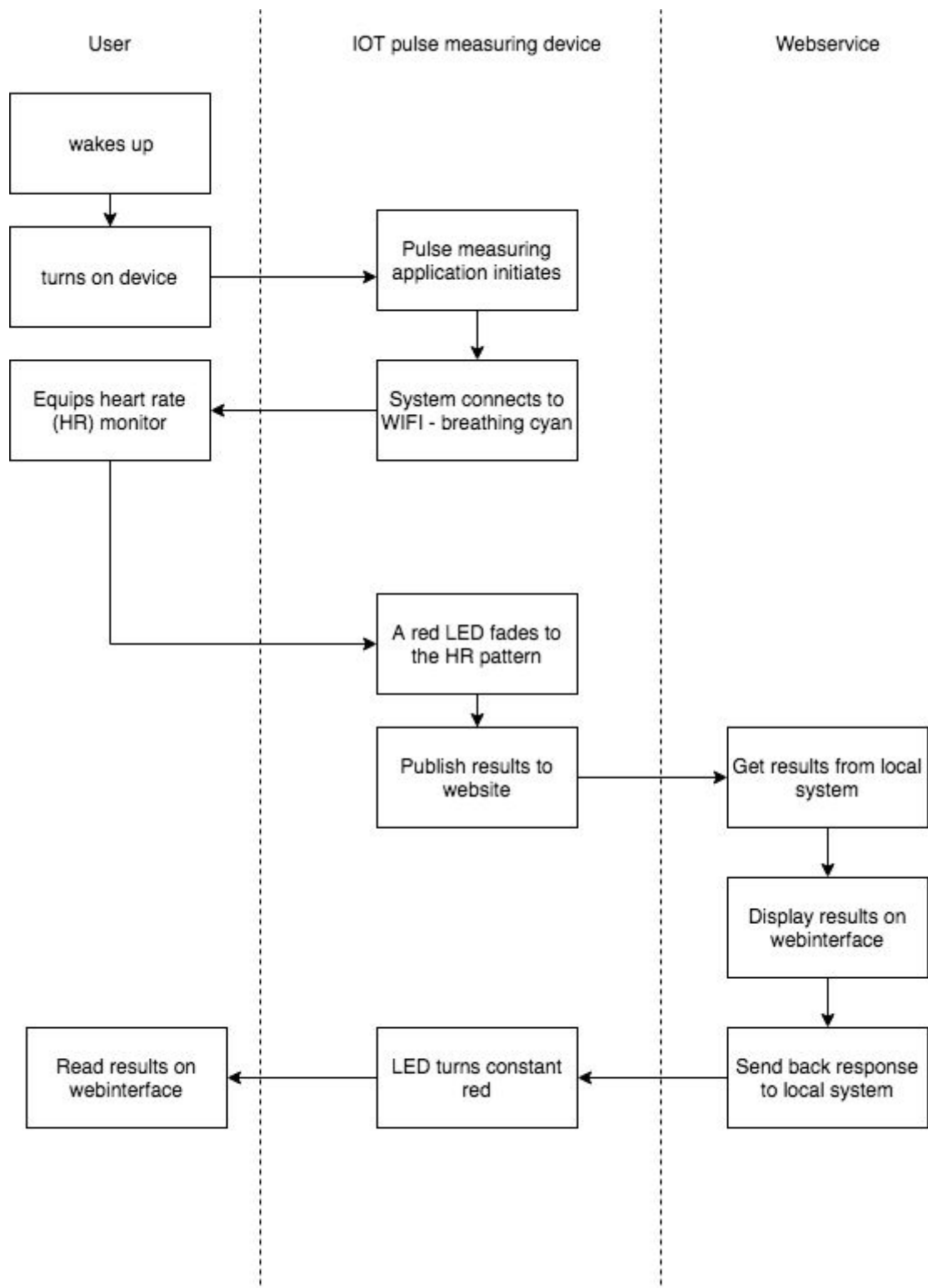

Figure 4. architecture of the system.

*Figure 5. use case diagram for the system*

Plan

Uge	Assignment	Deadline
35-37	Project selection Project description Purchasing hardware	14-sep
38	Interface analysis Presentation	21-sep
39	Setup development environment.	28-sep
40	Researching existing code for heart rate sensor.	5-oct
41	Implementation of heart rate sensor code	12-oct
43	Implementation of heart rate sensor code Presentation	26-oct
44	Get the whole heart rate measuring system working locally. User testing.	2-nov
45	User testing	9-nov
46	Send local data to webserver	16-nov
47	Response from webserver Presentation	23-nov

48	Integration test	30-nov
49	Integration test. Final adjustments, report	7-dec
50	Report Upload via wiseflow	14-dec

Implementation

A picture of the whole system is shown in Figure 6:
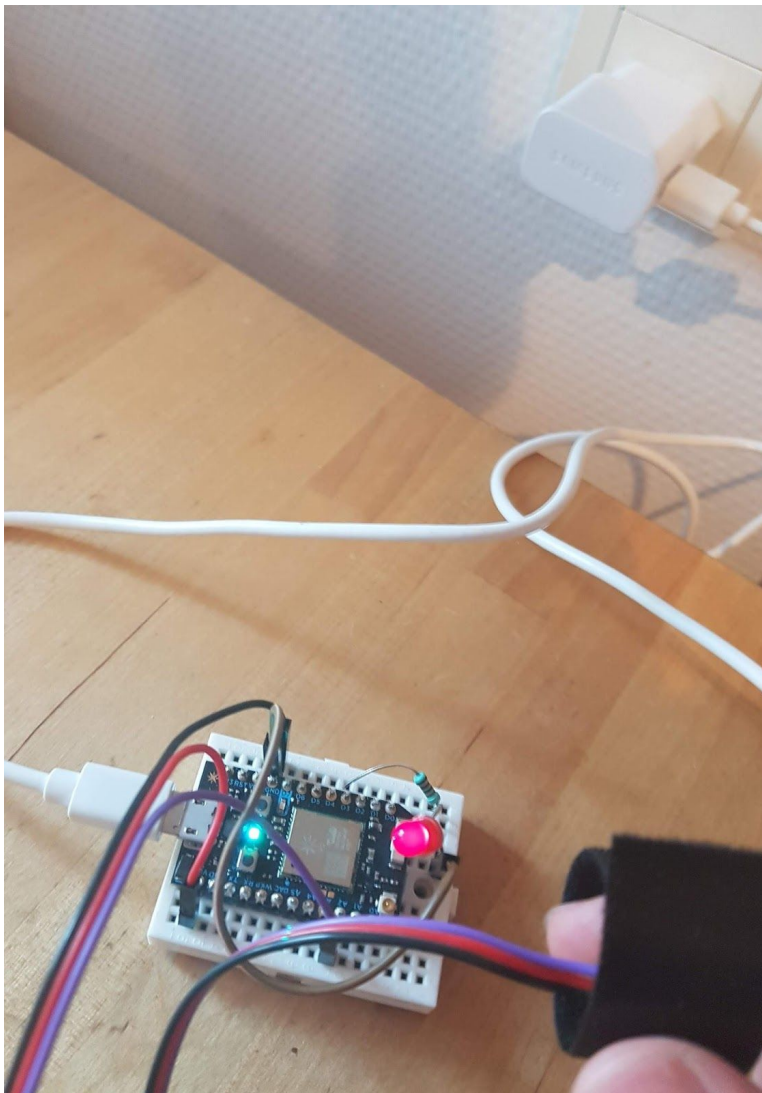
figure 6. The local heart rate monitor in use, plugged into a wall socket.

Software

The main source file for the project is "pulseIOT.ino. The dependencies are shown in the project.properties:

```
1    name=pulseIOT
2    dependencies.SparkIntervalTimer=1.3.8
3    dependencies.PulseSensor_Spark=1.5.4
```

The code is based on the "example" code from the following repository:
https://github.com/pkourany/PulseSensor_Spark.git
The code for the PulseSensor_Spark library can be seen in the "src" folder.

The SparkIntervalTimer library can be found here:
https://github.com/pkourany/Manchester_Library.git

The code for the whole project can be found here:
https://github.com/tobiasvalbjoern/IOT_project

The code for testing the communication between the particle photon, particle console, and thingsspeak webserver is found in hello_webhook.

General idea

First I thought about using a button to start the test. The code would be simple. The user would know when the test was started, and he could start it when he had found his pulse. The main downside to the idea was that I had to implement a push button, which added to the physical complexity. I want to keep the whole system very light. A push button is not directly related to measuring heart rate, and it is another thing that the user has to do, when he just wants to measure his resting heart rate.

Instead I wanted to make an algorithm to detect when the pulse has been detected, and when the readings are consistent. Every 2ms an analog reading of the sensor is done, and it is analysed in the background through the PulseSensor module. It is then determined if a heart rate has been found, and the BPM is found, and QS is set to true. It is these two variables that I use in my main logic. The logic for the pulseIOT.ino is shown in figure 7:
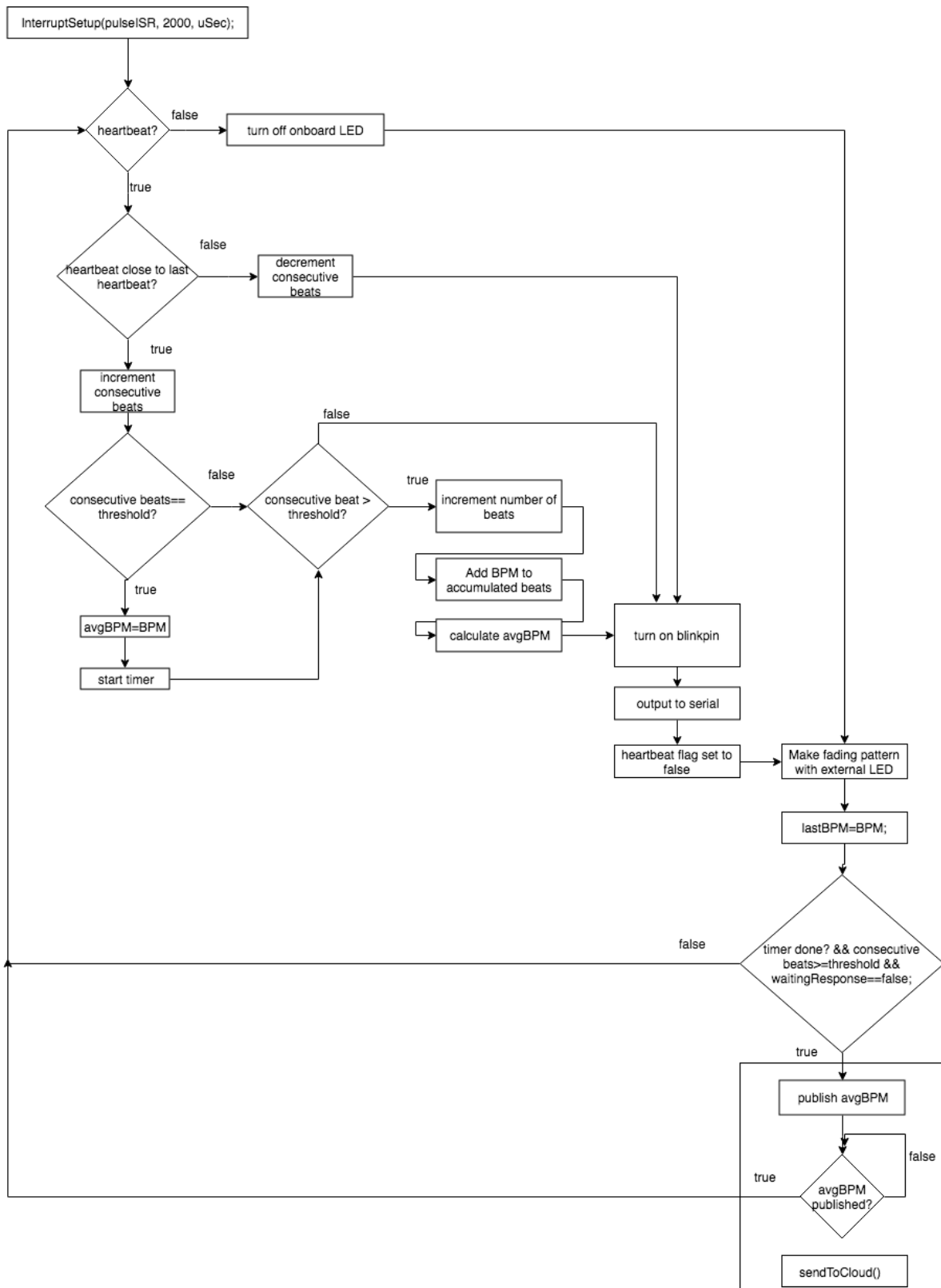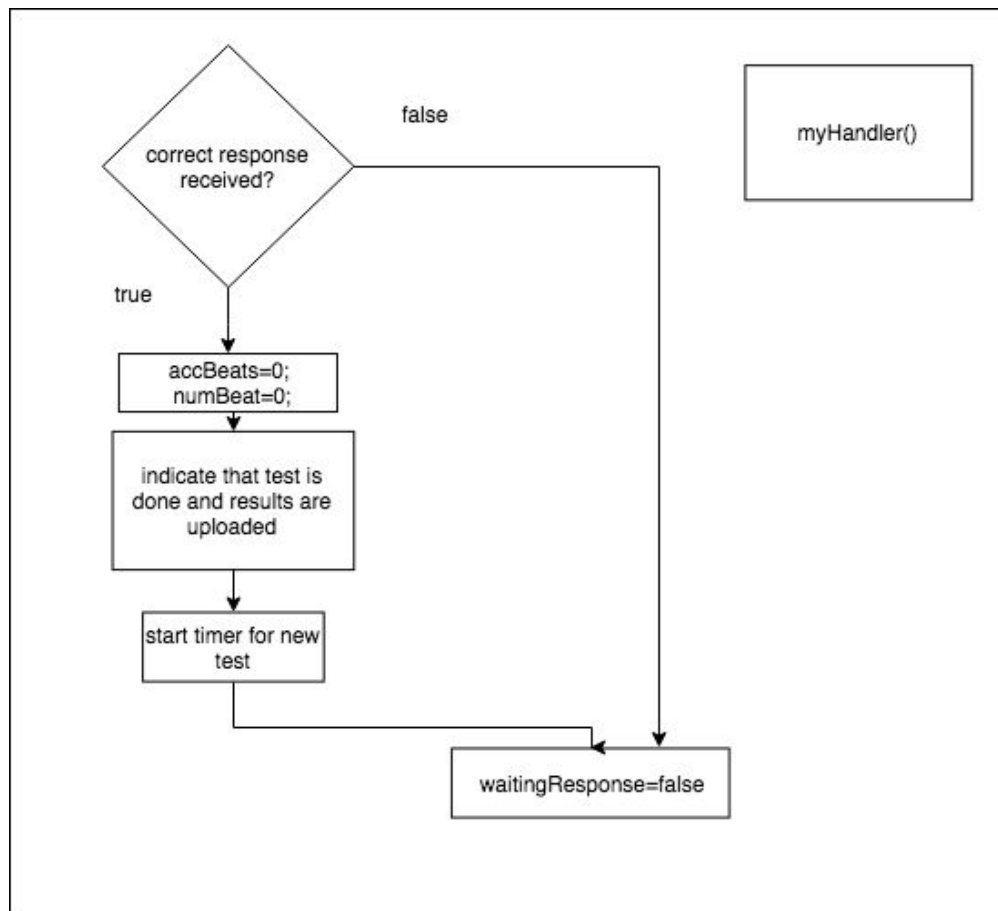
Figure 7. flow diagram of pulseIOT.ino

figure 8. flow diagram of myHandler(), the callback function for the subscription to the webhook to thingsspeak.

When you are first trying to measure your heartbeat with the heart beat sensor, it can be difficult. Often the sensor will make a few correct readings, and then the pulse disappears. Sometimes there is a big difference between each beat. The big red LED that is fading in a heart rate pattern helps you identify when your heartbeat has been found. I know from testing, that when the pulse has been found, the beats will be close to each other.

My first though was to count consecutive beat. Two consecutive beats is when two heartbeats are registered with a difference of less than 5 BPM. Another thing I noted, was that you could actually get several heartbeats in proximity before the pulse was found. Therefore I decremented the consecutive beats if the difference was greater than 5 BPM, and added a threshold. I experimented until I found a threshold of 10 to be enough. If I read 10 consecutive beats, each close the last one, it is most likely that the pulse has been found, and the test will start automatically. I did not want to upload several heartbeats to thingsspeak every time I took a test, I just wanted an average value, which would be most precise, since irregular heartbeats are averaged out.

When the test begins, the algorithm will calculate the average BPM for every heartbeat. When the test is done, the average BPM will be published to thingsspeak. The publish function will only be called once. I subscribe to the specific event name to get the response from the webserver, and I use a callback function called myHandler to handle the response.

NumBeats and accBeats are reset if the test results are successfully uploaded, a new test is started, if not, the BPM will be sent again. The reason for resetting the variables used to calculate the avgBPM, it that when doing multiple tests, the last test might be the most precise, because you have reduced your heart rate after resting, and therefore I don't want an accumulated error that builds up until that point.

Implementation of Thingsspeak

The procedure for setting up the connection between the particle photon and thingsspeak with webhooks was as follows:

1. Setup webhook on particle by following the guide on:
   https://docs.particle.io/guide/tools-and-features/webhooks/
2. Get particle device to publish "bpm" event.
3. Subscribe to the response from thingsspeak
4. Implement logic to show the user, that a successful response from thingsspeak has been received.

The code for publishing is shown below:

```
String data=String(avgBPM);
bool success=false;
while(!success){
    success=Particle.publish("bpm", data, PRIVATE);
 }
```

The event used in thingsspeak was "bpm".

The code used to subscribe to the webhook is shown below, from the setup():

```
//subscribe to the specific event name to receive the response
//from the web server and use it in the firmware logic.
Particle.subscribe("hook-response/bpm", myHandler, MY_DEVICES);
```

The callback function for the subscription is shown below:

```
void myHandler(const char *event, const char *data) {
/*
//event and data are C-strings. when comparing with "hook-response/bpm/0",
//a string object will be created. event=="hook-response/bpm/0" would
//never be true. Therefore strcmp has to be used.
 if(strcmp(event,"hook-response/bpm/0")==0 && strcmp(data,"0")){
  /*Serial.println("inside if statement");
  timerDone=millis()+10000;
  while(millis()<=timerDone){digitalWrite(fadePin, HIGH);}
  //Calculate the next time that data needs to be sent to cloud
  timerDone=millis()+10000;
 }
 waitingResponse=false;
}
```

When I published a message every 10 seconds, an error response would occur every second time. When sending a message every 20 seconds, a successful response is received every time. It appears that there is a response limit.

Test/verification

In the particle console all events can be seen, the published value, when the hook is sent, and when the response is sent. It is shown in figure 9:

Figure 9. events form particle console.

From the example shown, you can see that even from 20 seconds tests, the results are very consistent.

The results can also be seen on the account on thingsspeak as shown on figure 10.
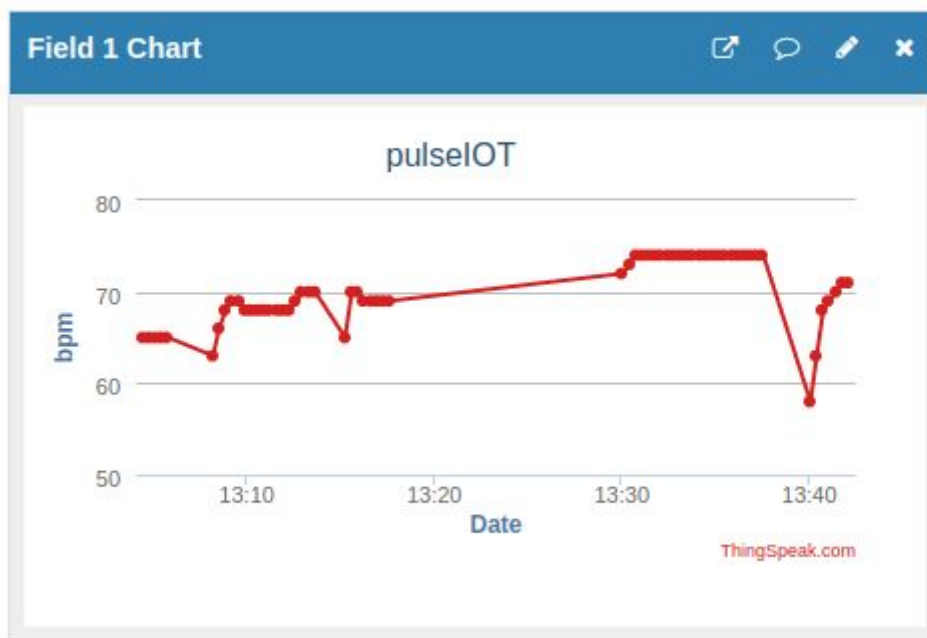

Figure 10. Heart rate on Thingsspeak, from the first few measurements.

One of the nice to have requirements, number 14, was that the whole system should be able to fit inside a box with the dimensions 15x10x8 cm. On figure 11 there is a picture on the system inside a box with the dimensions 15x10x5 cm:

Figure 11. Test 14. The whole system is in a lunch box with dimensions 15x10x5 cm.

An integration test has been made, and documented on video, and you can see it following this link:
https://youtu.be/-vHqDUqJhV4

I observed that it took more time than anticipated to find the pulse, so I took 10 tests, and calculated the average time:

Start up 5 seconds
Test of startup time.
1. 1min35sec
2. 1min13sec
3. 20 sec
4. 52sec
5. 20sec
6. 36 sec
7. 12 sec

8. 17 sec
9. 13 sec
10. 1min10sec

Average time: 40.8 sec

Requirement no.	Requirement type	Comments	√
1. The platform has Wifi connectivity, and the system will connect over Wifi.	Need to have	See the video that shows the integration test.	√
2. When the Wifi has been set up, the system should automatically connect to the internet when power is turned on.	Need to have	See the video that shows the integration test.	√
3. After power is on, the application should start running automatically.	Need to have	See the video that shows the integration test.	√
4. The platform has available digital or analog I/O to connect sensors and actuators.	Need to have	See the video that shows the integration test.	√
5. The system reads the resting heart rate from a heart rate sensor local to the device.	Need to have	See the video that shows the integration test.	√
6. The system should be able to send the heart rate locally from the particle photon to thingsspeak.	Need to have	See the video that shows the integration test.	√

7. The device controls an actuator in the form of an LED.	Need to have	See the video that shows the integration test.	√
8. The device uses data from thingsspeak to augment what it does, by giving the user feedback on the LED	Need to have	See the video that shows the integration test.	√
9. The webservice should be able to show the results of the heart rate test on thingsspeak.	Need to have	See the video that shows the integration test.	√
10. The system should be able to measure the heart rate accurately. Within 2 bps difference from a Garmin Vivosport wrist heart rate monitor.	Nice to have	See the video that shows the integration test.	√
11. The whole system should be able to fit into a box with the maximum dimensions of length 15 cm, width 10 cm, height 8 cm.	Nice to have	See photo of system inside lunch box with dimensions 15*10*5 on figure 11.	√
12. The system should be able to store the necessary information of 1 user from the website. (age, sex, previous heart rate	Nice to have	Not implemented	x

measurements, category, and date)			
13. The webservice should be able to compare the heart rate data with previous data.	Nice to have	Not implemented	**x**
14. The webservice should be able to compare the heart rate data with the heart rate table.	Nice to have	Not implemented	**x**

Non-functional requirements

The non-functional requirements are harder to put in a table and check off. They have been used as general principles when making decisions. I will go through each of them.

The system needs to be simple, with as few user interactions as possible.
The system functionality will be focused on the task of measuring the resting heart rate.

I have returned to these requirements several times. It was used in the decisions with RGB, LCD-display, vibrator motor, and button. It helped me make the decision of staying with a simple red LED. All of the considered actuators or sensors could be nice to have in a heart rate monitoring system, but in the end they were all features, not part of the core functionality of the system. I wanted to focus on what the system does best, measuring the heart rate.

Existing standards and protocols will be used wherever possible.

When I was first starting out there were decisions to be made about where I should use existing solutions, and where I should make my own. I used this requirement to chose code for the project that detect a heartbeat every 2 ms, and an example of how to utilize it. I then implemented my own logic using that framework. This choice has helped me get to the point where I have a whole "IOT chain" from local sensor, to webservice, and back out through an actuator. If I wanted to make my own code to analyze the analog sensor from the signal, I might not have gotten farther.

The system will be calm when everything is working fine. It will not bring attention to itself beside when giving the user instruction or feedback.

The system has built in LED's to show the status of the system, ie when starting up and connecting to the internet. When it is ready to measure the pulse, the built in LED or external RED LED does not do anything. This is a conscious choice based on this requirement. As a consequence, it is very clear, when you strap on the monitor, and the LED's start blinking in the pattern of your heart rate. Another choice I have made, is when the tests starts. There is no indication. I thought about making it clear to the user that the test starts, but based on this requirement, and on the requirement to keep it simple, and use as few user interactions as possible, I found that it was not necessary. The user can just assume, that when the pulse has been found, everything is working as it should. The most relevant information for him, is when the test is finished, and the results are uploaded, and he gets a clear signal when that has happened.

Discussion

The idea was to improve and automate the process of keeping track of the resting heart rate. The project was meant to be a cheaper alternative to the wrist heart rate monitor.

During the project I bought a fitness tracker, a Garmin Vivosport, with a wrist heart rate monitor. In that way, I can accurately compare my project with the intended product. First of all my project is a lot cheaper. I have spent a little under 200 kr. for the hardware plus transport. The wrist heart rate monitor costed 1100 kr. My project is 5 times as cheap.

My project was never intented to be better than the wrist heart rate monitor, it was more thought of as a "poor man's fitness tracker". My Garmin Vivosport tracks my resting heart rate while I am sleeping, and I will never be able to do that with my project. Even if I use the system immediately after waking up, my pulse will still be around 47, compared to 43 while asleep. Furthermore, the Vivosport tracks my heart rate automatically, and my project cannot do that either.

Lastly, it takes some time to adjust my heart rate system, sometime a couple of seconds, and a few times even more than a minute. The average time was found to be around 40 sec. That might be the biggest drawback, and it is something that could be added as a possible improvement of the system.

When the heart rate monitoring system is correctly applied, and the pulse has been found, the uploaded results are very accurate. Taking all that into consideration, I would never use my system, when I have my fitness tracker, but if I could not afford a fitness tracker with wrist heart rate monitoring, I would definitely use my IOT pulse system, and I would be confident in the accuracy of the results. The system can fulfill its purpose of measuring the resting heart rate over time, in order to evaluate your general fitness.

All the "need to have" requirements have been met, and many of the "nice to have" requirements have been met too. It is the advanced web interface that has not been

implemented, where a user would be able to store information, see fitness level, and improvement.

Conclusion

The overall purpose of the project was to apply theoretical topics from the course onto a specific project. The project was to design an IOT artefact with embedded hardware, software, sensors, and actuators, and the IoT artefact should be able to perform a meaningful task better than a similar device not connected to the internet.

The core functionality of my project is measuring the heart rate, uploading it to thingsspeak, receiving the response from thingsspeak, and then telling the user, that everything went successfully. I have compared my system to another system that is also an IOT device, my fitness tracker, but if i compare it to another device that is doing the task, that is not connected to the internet, it will be the chest strap, along with a watch. In my opinion my system does that task a lot better than the chest strap monitor. I have never used my chest strap monitor + watch once to calculate my resting heart rate. I do not want to sleep with the chest strap on, and even if I did, it would not be moist enough. Furthermore, if i apply the chest strap+moist upon waking up, it is not my resting heart rate that I am measuring. In that regard, the project was successful.