AARHUS UNIVERSITET

Aarhus School of Engineering, Herning

Smart Doorbell-IOT 2018

Authors:

Kim C. Nielsen (stud.nr: 201608167)

Mads Villadsen (stud.nr.: 201609607)

Lasse A.Frederiksen(stud.nr.: 201606615)

Supervisors:

Morten Opprud Jacobsen

Klaus Kolle

Table Of Contents

1 Introduction (Kim)

This report documents the 5th semester Internet Of Things project by Mads H. Villadsen, Kim C. Nielsen and Lasse A. Frederiksen. The project focuses on solving the problem about how a smarter doorbell, by the aid of electronics and internet technologies, can help an ordinary resident become more knowledgeable about visitors.

The reader can expect thorough and detailed descriptions for the entire development process from idea to test and verification of each requirement. This report and source code is publicly available on Github. [3]

2 Project Description (Lasse, Kim, Mads)

Our project idea is based on a doorbell with an attached camera. The idea is that if someone is using your doorbell you will get notified by a mail notification. The mail will contain an image of the person who used the doorbell. This way you get an exact idea about who used your doorbell.

The system can also be used in a case, where you are not interested in answering the door. This might be the case, since the person using the doorbell is someone you aren't interested in talking to at the moment etc.

If you happen to be occupied by listening to music via for instance Spotify, the doorbell system can pause the music, in order to notify you about someone using the doorbell.

The doorbell system integrates events from your calendar system, meaning that a speaker will notify the person pushing the button that you are either busy or available. A further development solution of this would be to have it only happen if it was a person known by the system.

3 Requirement Analysis

This section describes the process of capturing the requirements for the system that is to be developed.

The number are not an order of priority but a tool to make the differentiation of the requirements easier.

3.1 Need to have Requirements (Lasse, Kim, Mads)

Supply

3.1.1 The device outside shall should run off batteries

3.1.2 The device inside shall run on ordinary supply

Consumption

3.1.3 The device in deep sleep mode shall only draw a maximum current off 100uA

3.1.4 The device should have a large enough battery capacity to keep it running for a month without changing batteries

Device Inside

3.1.5 The device inside shall have a speaker able to play music by a press on the button

3.1.6 The device must be able to integrate online calendar system, so the other speaker can inform if the owner is busy or not. Online calendar can be e.g. google calendar.

3.1.7 The device will be able to pause music/video from e.g. spotify or youtube etc.

Device Outside

3.1.8 The device outside shall have a button attached to wake up the board

3.1.9 A camera needs to be attached so that when the board wakes up it will take a picture

3.1.10 The picture needs to be send by email to the owner

3.2 Nice to have requirements (Lasse, Kim, Mads)

3.2.1 The device will have a speaker to inform the visitor that the owner is busy.

3.2.2 The device outside needs to be water resistant

3.3 Technical Requirements (Lasse, Kim, Mads)

3.3.1 The platform needs to have Wifi connectivity.

3.3.2 The platform shall consist of GPIO pins for analog and digital connection purposes to sensors and actuators.

3.3.3 The platform shall be able to operate in deep sleep mode.

3.3.4 The camera needs to be able to take pictures at a minimum quality of 480x360.

3.4 Technical Platform Specification (Lasse, Kim, Mads)

3.4.1 Development board

Based on the technical requirements, a specific platform has been selected.The selected platform for development in this project, is a Photon Particle Board. This board has been chosen, since it is cheap, compact, low power and features Wifi access via a ESP8266 Wifi chip. Another important parameter is the size. Because of it’s small size it can fit practically anywhere. The board also has plenty of IO ports compared to its small size. [1]

3.4.2 Programming language and IDE

Particle Build & Atom:
Firmware for the Photon board can be written in a web IDE called Particle Build or a local editor such as Atom. The IDE includes all necessary tools in terms of a compiler, library manager, serial monitor etc. [2]

C++:
C++ is an object oriented programming language, which means that it is structured through classes and objects. The photon board uses C++ code with a setup like an arduino board so that you will be able to pull from arduino resources.

5 System Design

5.1 System diagram (Lasse, Kim, Mads)

The overall system diagram is visualised on figure 5.1.1. In order to distinguish between the system components, the hardware components are colored in grey and software components are colored in blue. The photon boards communicates with both software and hardware, which is why both boards are colored in purple. Data transmission is illustrated with lines. The black solid lines illustrates a wire and the dashed lines illustrates Wi-Fi connectivity. The Blue solid wires can be thought of as virtual wires between the photon boards and the software components.

The Photon board 1 is located outside and is connected to a VC0706 camera, speaker and a button. If a person decides to push down the button it will trigger an input signal, which can be received by the Photon board 1. On the basis of the input signal, the photon board will ask the camera to take a picture. The picture will subsequently be uploaded to Google drive, which will trigger an event. Based on the event, it is acknowledged by the system that a new file was uploaded and it will be attached inside a mail and the photon board will send it to the owner. The speaker is supposed to play a specific sound that informs the visitor about the owners whereabouts. Lastly, the photon board 1 will make the speaker play if the photon board 1 registers that the owner's Google calendar system contains a current event that has not ended yet.

The Photon board 2 is located inside and is connected to a speaker. This speaker will play a sound like a normal doorbell, once the button outside is pushed. If the owner is currently listening to music on Spotify, the photon board 2 applies the Stop Spotify music software component to pause the currently playing music.

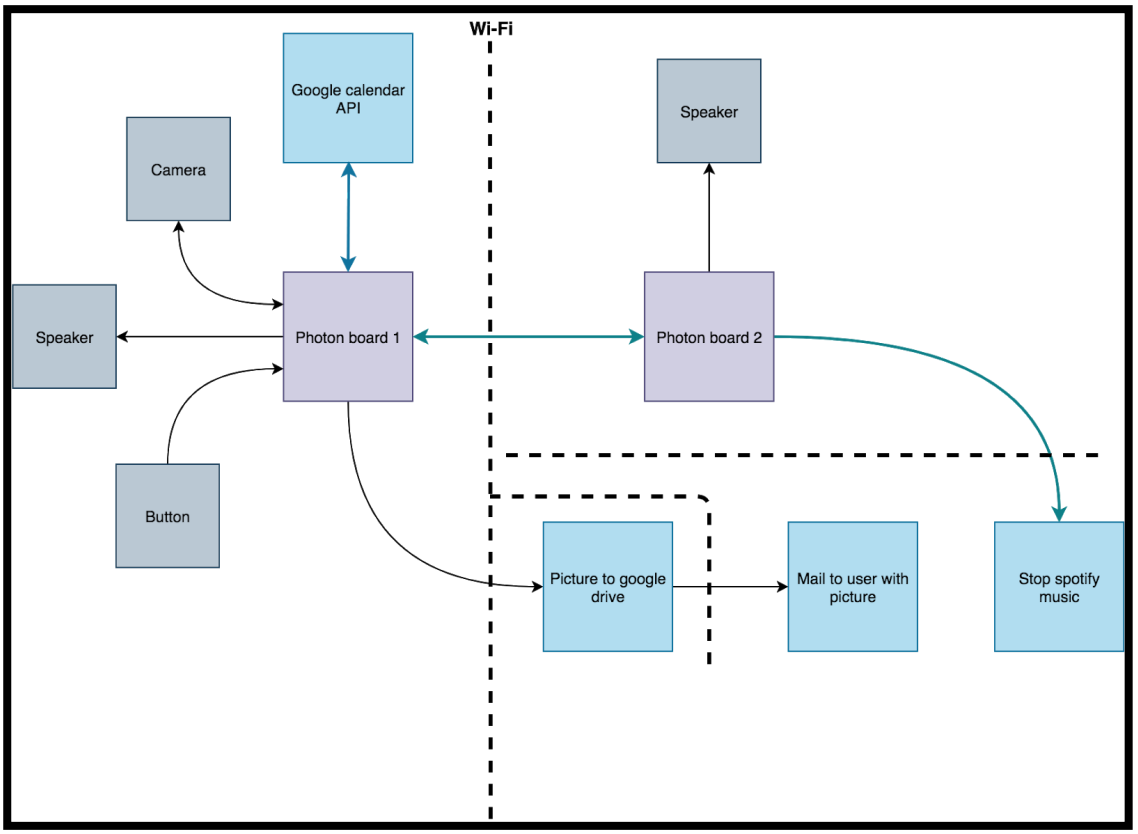

Figure 5.1.1 - architecture diagram

5.2 Power Design (Kim)

Power consumption is a crucial part of our system, since our doorbell system outside can be considered a low-power device. This is the case, since the doorbell system outside is powered by a 9V battery. Wireless doorbell systems typically has a battery lifetime that would last for about a year. However, the temperature can influence the battery life. For instance cold weather can shorten the battery life. [7] However, since our doorbell system is also controlling a camera and a speaker, it automatically needs power optimization, in order to decrease the power consumption and insure that the battery lifetime of the system remains acceptable.

Power optimization can be achieved by several different techniques. However, the chosen power optimization technique depends on what purpose it serves. In our case we want the doorbell system outside to operate in deep sleep mode most of the time.

The technical platform specification section mentioned that the applied development board had been selected to be a Particle Photon board, since it features, among other things, low power characteristics. Measurements of power consumption for the device outside has been collected, as the device operated in the different states. The results can be seen on figure 5.2.1 in the Power Profile section.

Power Profile

The Particle Photon board contains a STM32f205RGY6 MCU, which supports three low power modes. The low power modes are sleep mode, stop mode and standby mode. [12]

The sleep mode results only in the CPU being stopped. As a result, the peripherals will still continue to operate and they can wake up the CPU using an interrupt/event. Using this low power mode with the Photon board will also include deactivation of the Wi-Fi module, which is also responsible for the biggest power consumption. [11, 12]

The stop mode results in what is known as very low power mode. This mode will still keep the contents within the SRAM and registers, but all clocks in the 1.2V domain are stopped, HSE crystal oscillators are disabled etc. Using this low power mode with the Photon board will also deactivate the Wi-Fi module, pause the CPU and the running application. However, a pin can be used, so that an application is able to resume. The only requirement for the pin is that it is either D1-D4 or A0, A1, A3, A4, A6 or A7. These pins can supports external interrupts. [11, 12]

The standby mode, which is also known as deep sleep mode, results in what is known as ultra low power. It is somewhat similar to the stop mode, but instead of keeping the contents within the SRAM and registers it doesn't keep anything at all. Using this low power mode with the Photon board will turn off the microcontroller and the Wi-Fi module. [11, 12]

We have tried out two of the mentioned low power modes for the Photon board sitting outside, and the results can be seen on figure 5.2.1. The figure indicates a table with three different states and their corresponding values in terms of current. The stop mode resulted in a current of 2mA, which is significantly lower than the current that occurs under normal operation, which is 71mA. The best performance in terms of the lowest current was during the Standby mode that resulted in a current of 114 uA. We did not measure the current during sleep mode, since there is no reason to keep the application running constantly.

State	**Current [mA]**
Stop mode (Very Low power mode)	2 mA
Standby mode (Ultra Low power mode)	114 uA
Normal operation	71 mA

Figure 5.2.1 - Power Profile of outside device

6 Implementation

6.1 Mail (Lasse)

To send mails we use an IFTTT applet as well as the particle photon board the in depth of the applet can be seen on figure 6.1.2, 6.1.3, 6.1.4, 6.1.5, 6.1.6 and 6.1.7. On figure 6.1.1 the IFTTT applet for sending a mail where if the particle board called The_Doctor publishes a SendMail event it will send an email to iotmadskimlasse@gmail.com. The_Doctor is the name of one of the two photon boards that we will use the other is called Lasse-Photon-Torpedo.

Figure 6.1.1 - IFTTT of the mail sending applet

On figure 6.1.2 the first thing that we have to do to set up an IFTTT applet can be seen where we will choose if this then that, and here we will start with choosing the trigger by clicking on the blue this from there we will be able to choose from over 600 different services.

Figure 6.1.2 - Create a new applet and choose services

On figure 6.1.3 a few of the services available from IFTTT can be seen where we will be using the Gmail service, the Particle service and the Google Calendar service, where the Gmail service will be used to send mails, the Particle service will be used to publish events to send a mail and subscribe to events from the Google Calendar service, which will be explained further in section 6.2.

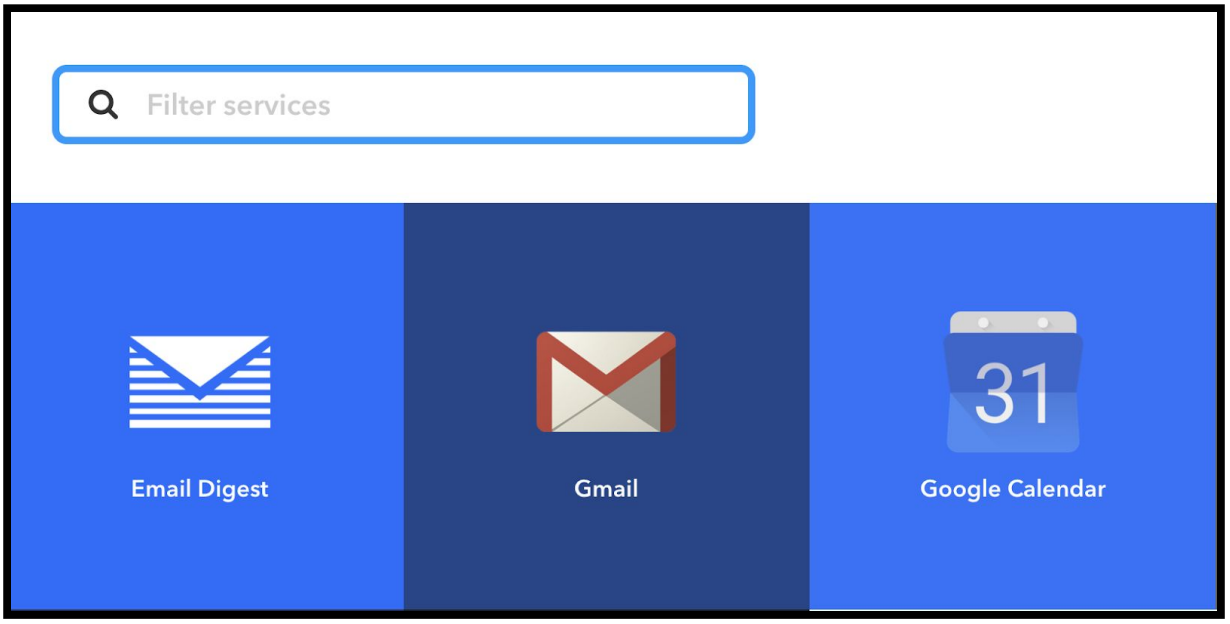

Figure 6.1.3 - Example of services that can be connected to IFTTT

On figure 6.1.4 the different triggers for the particle service can be seen, where we can choose from triggering when a new event is published set it to monitor a variable, monitor a functions result or to monitor the states of the device. For our purpose we will be using when a new event is published

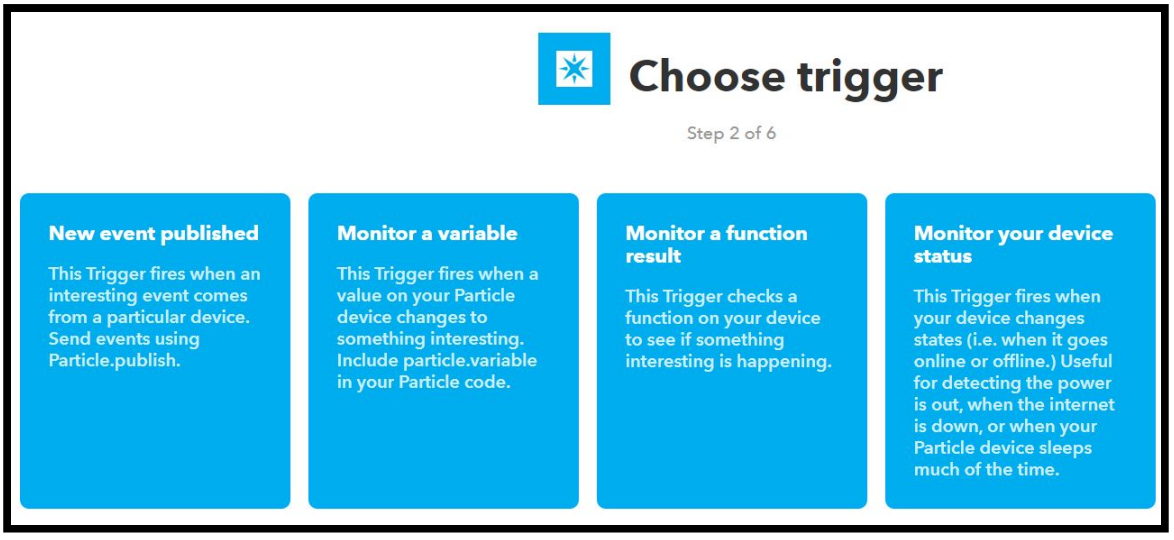

Figure 6.1.4 - Setting up the trigger from for an IFTTT with particle

On figure 6.1.5 the event form the particle board that will trigger the IFTTT applet can be seen here. The name of the event is SendMail and the device that we will be using to publish this event is called The_Doctor.

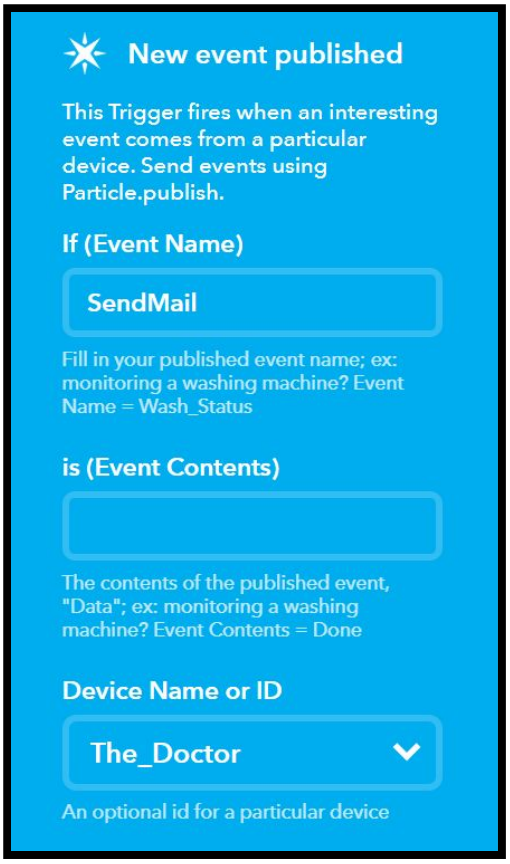

Figure 6.1.5 - Setting the event that triggers IFTTT

On figure 6.1.6 the different actions that are available to Gmail can be seen where it can send an email to a up to twenty recipients send yourself an email or create a draft of an email that you would be able to send later yourself. For us we will be using the send yourself an email action since that way it will be easier for us to test and it could be further developed, which will be discussed more about in section 9.

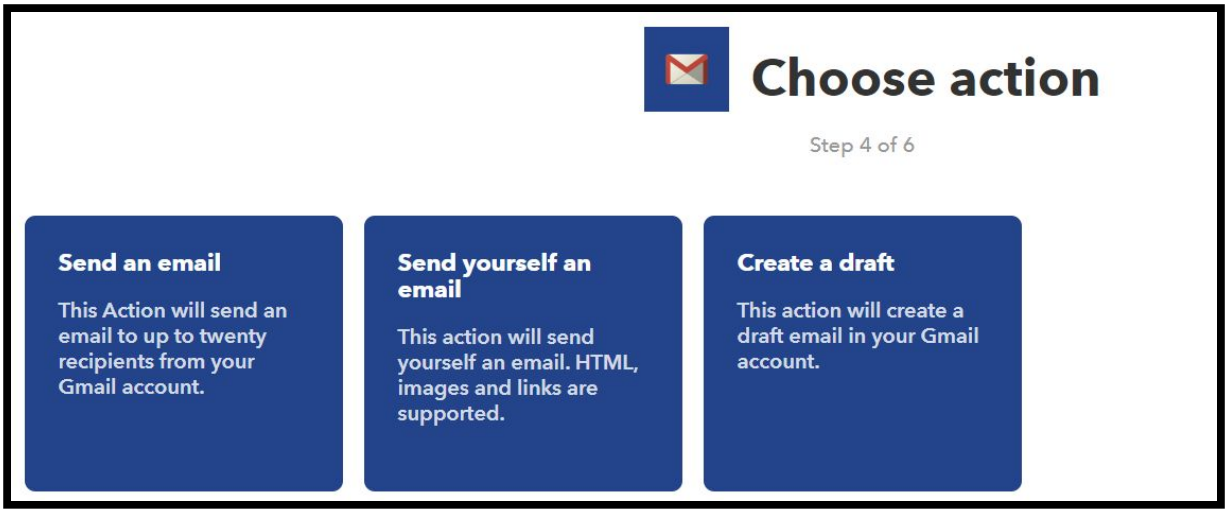

Figure 6.1.6 - Choose action for gmail applet

On figure 6.1.7 the action that the Gmail service will take can be seen where it will send itself an email with the subject “Doorbell” to tell the user that it is doorbell that just emailed them and with the body “Somebody is at the door“ to explain the user that somebody has just pressed the doorbell button. The reason that currently there is no attachments setup is because this is where the picture that the camera took was to be setup and more about the reason why can be read in section 7.4.1.

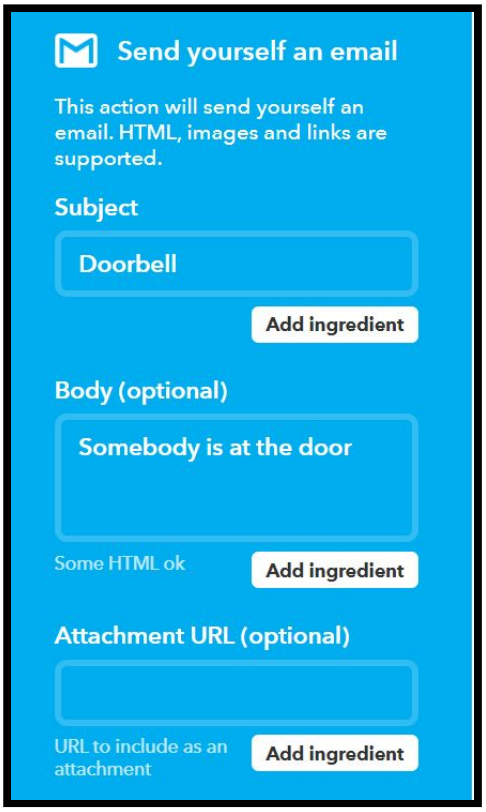

Figure 6.1.7 - Setting up the action

To get the particle photon board to trigger the event that will make the applet send a mail we use the command Particle.publish, which can be seen on figure 6.1.8, which can take either a string with the event name or both the event name string and a string with data in it, the particle publish will return a boolean, true if it was successful with publishing the event and data if any or false if it is not successful in publishing the event and data.[1]

```
// send mail to user
Particle.publish("SendMail");
```

Figure 6.1.8 - Publish SendMail

6.2 Google Calendar (Kim)

Smaller parts that makes up the implementation of the Google Calendar component is illustrated on figure 6.2.1, figure 6.2.2 and figure 6.2.3. On figure 6.2.1, a boolean variable and the setup function is shown. The boolean variable homeState is used to define if the user is home or not. The variable is initialized to be true, which means that the system initially assumes that the owner is home. The setup function is setting up subscriptions for two events that is Start_Event and End_Event as well as their belonging event handlers. The subscriptions sets up the two events because when the particle uses the Particle.subscribe command it needs as the first value the name of the event as a string and the last is the handler for the event, which needs to be a function.The Start_Event and the End_Event are applets that are connected with Google Calendar through the IFTTT service. The applets can be seen on figure 6.2.4.

```
1   bool homeState = true;
2
3   void setup()
4   {
5       //  Subscribe to IFTTT events:
6       Particle.subscribe("Start_Event", startEvent);
7       Particle.subscribe("End_Event", endEvent);
8
9       Serial.begin(9600);
10      Serial.println("Connection established!");
11
12  }
```

Figure 6.2.1 - setup function

The event handler for Start_Event can be seen on figure 6.2.2. The event begins by evaluating if the name of the event is ‘Start_Event’ using the strcmp function. If the strings are queal, then the function returns 0 and the event handler proceeds to set the homeState variable to false, indicating that the owner is not home. The Start_Event handler will run if any Google Calendar event starts.

```
//  Event handler for started events:
void startEvent(const char *event, const char *data)
{
    if( strcmp(event, "Start_Event") == 0 )
    {
        Serial.println("Event started!");
        homeState = false;
    }
}
```

Figure 6.2.2 - Event handler for started events

The event handler for End_Event can be seen on figure 6.2.3. The event handler is similar to the event handler for Start_Event. The only difference is that the homeState variable is set to false, indicating that the owner is home. The End_Event handler will run if any Google Calendar event ends.

```
//  Event handler for ended events:
void endEvent(const char *event, const char *data)
{
    if( strcmp(event, "End_Event") == 0 )
    {
        Serial.println("Event ended!");
        homeState = true;
    }
}
```

Figure 6.2.3 - Event handler for ended events

The IFTTT applets for Start_Event and End_Event can be seen on figure 6.2.4. It can be seen that the applets are connected to a Google Calendar that belongs to the Google account iotmadskimlasse@gmail.com. If any event starts or ends on the calendar, an event will be published and the Photon board outside can react.

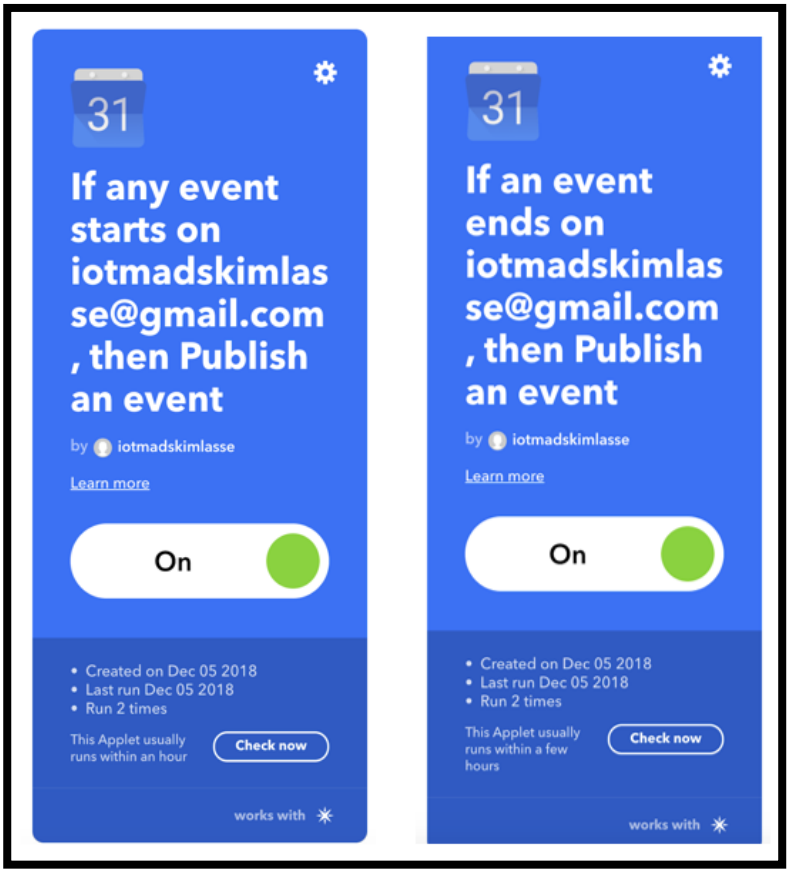

Figure 6.2.4 - Start_Event and End_Event Applets

6.3 Spotify (Lasse)

To pause the currently playing track of the user we looked at the spotify API documentation to see if they had any api functionality, which could be used for what we needed and there we found the pause functionality, which can be seen on figure 6.3.1. It can be seen that it needs to have an authorization token send and according to the Spotify api, a premium user account is needed to be able to use the API functionalities. Luckily, one of our group members (Mads) was already paying for several premium accounts, where some of them were not used so we set up a test account to be able to use the Spotify api functionalities.[4]

Description	Pause a User's Playback DOCS
Endpoint	https://api.spotify.com/v1/me/player/pause
HTTP Method	PUT
OAuth	Required

Figure 6.3.1 - Spotify api document on pause functionality

Since we need to use authorization tokens we need to be able to have access to the authorization flow which can be seen on figure 6.3.2. It is here seen that first the program should be able to request the authorization where the user would have to log in and that way we would be able to get the authorization and refresh tokens so that we could keep the access to the pause functionality otherwise the authorization token will only last for 1 hour[5].

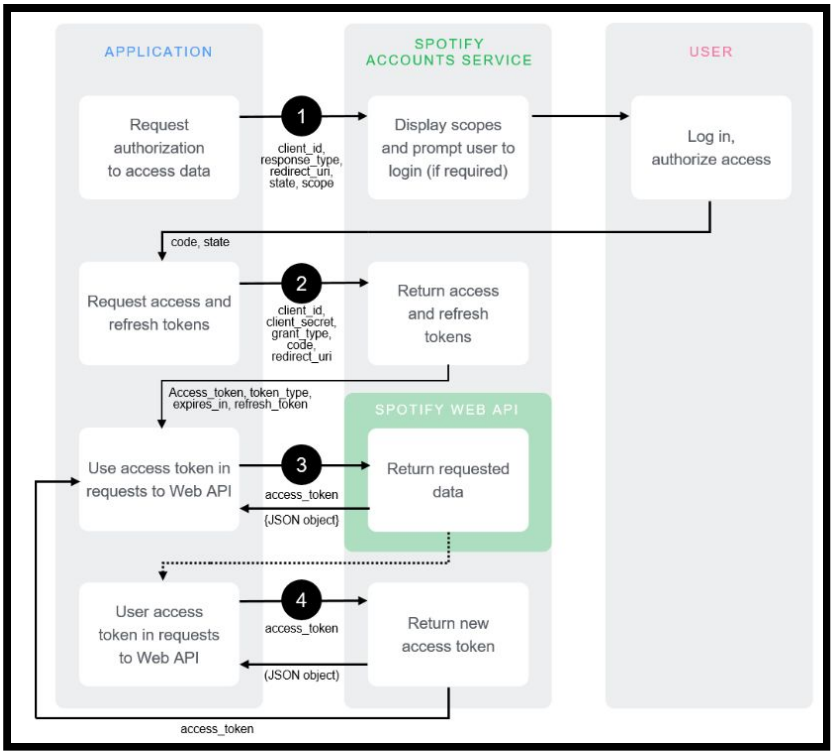

Figure 6.3.2 - Authorization code flow

On figure 6.3.3 the integrations that have been implemented and the way to create a new integration can be seen. We have created a webhook that we will use to pause the music playing on Spotify. A webhook works in the way that it will send a HTTP or HTTPS request to a specific website here the pause endpoint from figure 6.3.1.

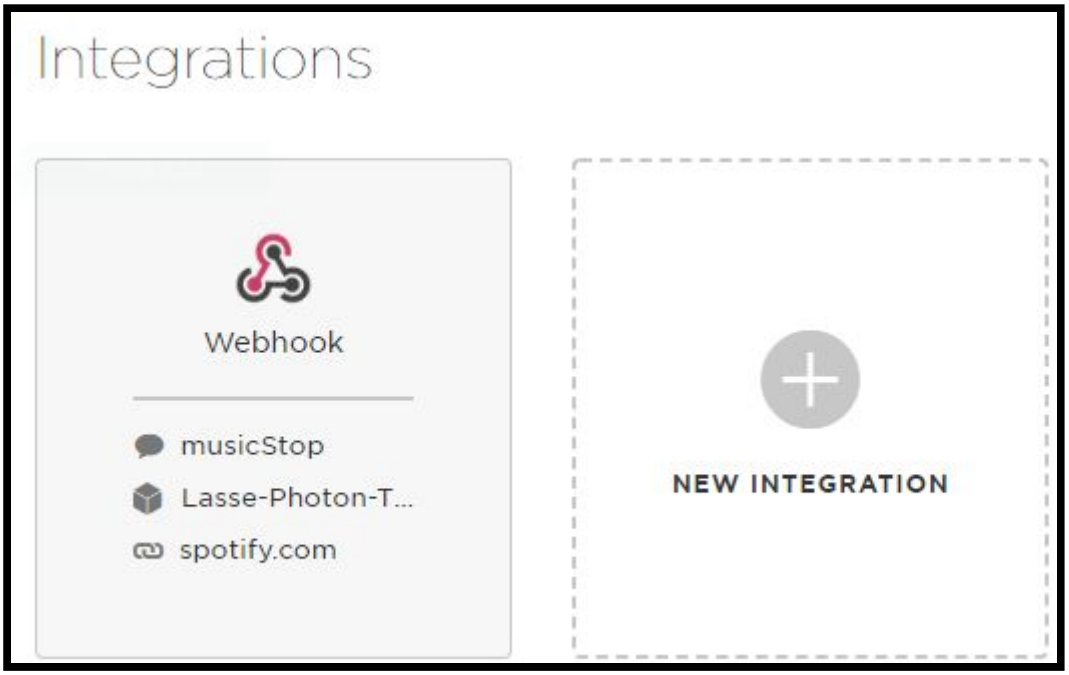

Figure 6.3.3 - Webhook for Spotify

On figure 6.3.4 we can see what kinds of integration we are able to set up where we will be using a webhook.

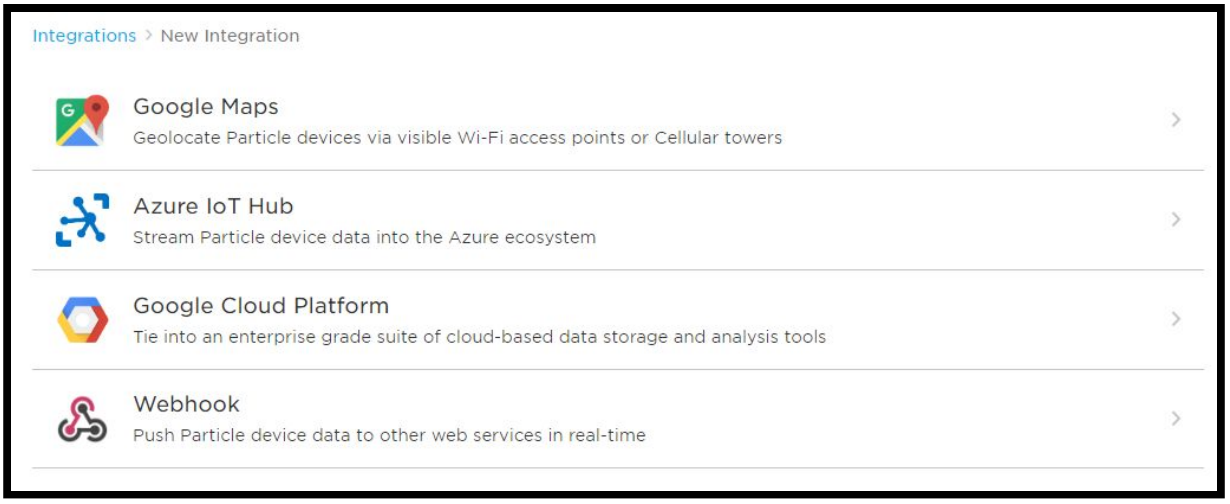

Figure 6.3.4 - Integration types for Photon

On figure 6.3.5 the webhook builder can be seen, where it will trigger on the musicStop event it will send a request with the type PUT to the endpoint URL from the Spotify API on figure 6.3.1 the request format is set as a JSON object and the device that will trigger the musicStop event is set to be The_Doctor.

Figure 6.3.5 - Webhook settings

On figure 6.3.6 the id of the device running Spotify will set up so that it is a specific device where spotify will be paused. We set in the authorization token in one of the headers as well. Unfortunately setting up an automation for gathering the authorization code and integrating it with the webhook shown has not been possible.

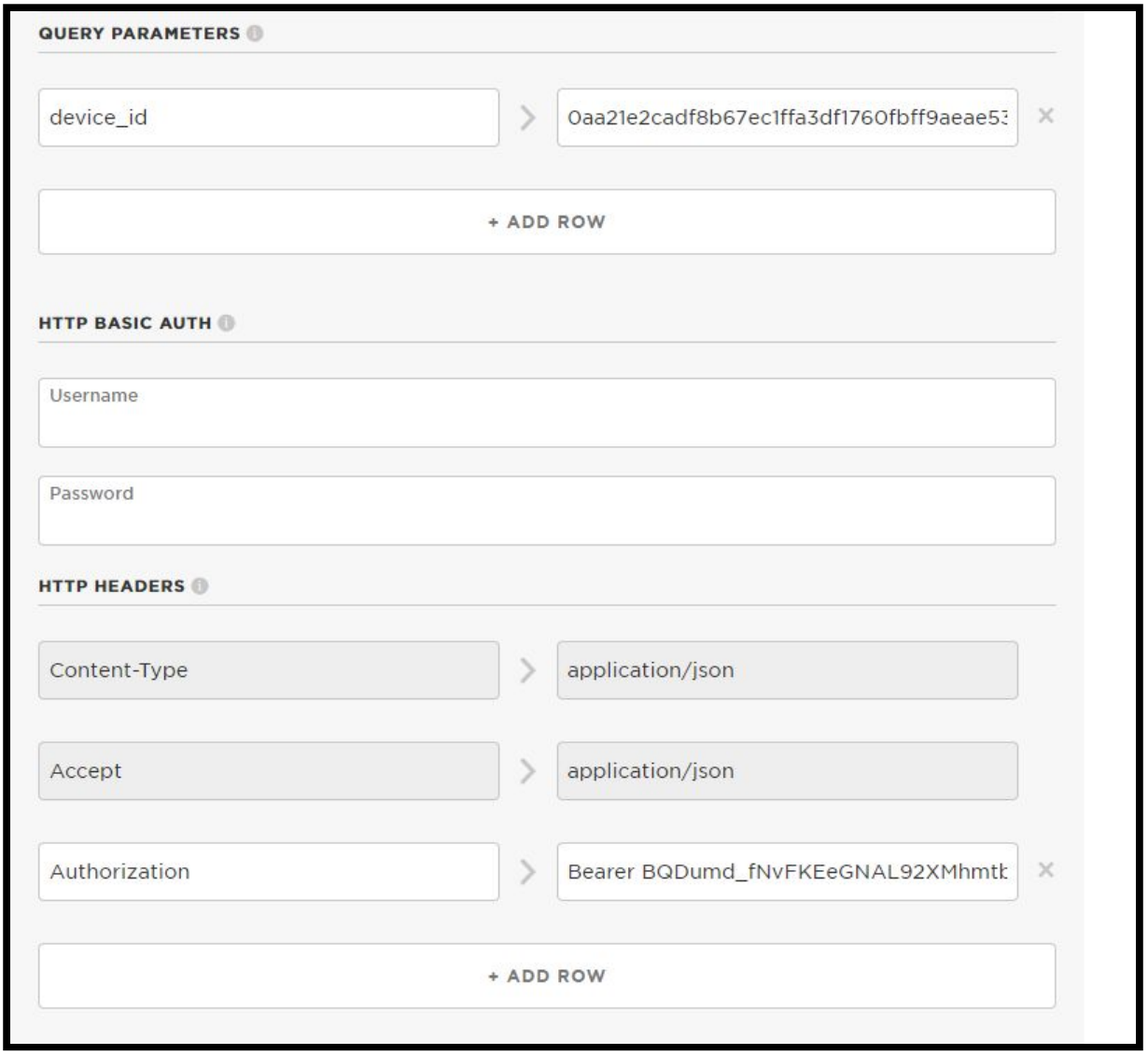

Figure 6.3.6 - Http request settings

On figure 6.3.7 same as is shown in figure 6.3.5 and 6.3.6 can be seen just in a JSON format instead of the webhook builder layout.

```
{
    "event": "musicStop",
    "deviceID": "3d001d000c47363433353735",
    "url": "https://api.spotify.com/v1/me/player/pause",
    "requestType": "PUT",
    "noDefaults": false,
    "rejectUnauthorized": true,
    "headers": {
        "Authorization": "Bearer Authentication-token"
    },
    "json": true,
    "query": {
        "device_id": ""
    }
}
```

Figure 6.3.7 - Webhook custom template/data form

6.4 Camera (Kim)

The wiring of the Photon board and the camera can be seen on figure 6.4.1. The supply voltage is taken at the Vin pin and used to supply the camera. When the Photon board is connected to a USB port, the vin pin will output a DC voltage of 4.8V. It can also be seen on figure 6.4.1 that a pair of digital pins are used to connect to the TX and RX pins on the camera. The used digital pins on the Photon board are D2 and D3. The wiring is inspired by a tutorial from Adafruit.[1] The tutorial shows how to use the camera with an Arduino board, but since the Arduino board is able to output a voltage of 5V from any digital pin, it is required to use a voltage divider with a pair of 10K resistors. The voltage divider ensures that the voltage will not exceed 3.3V. 3.3V also happens to be the highest output voltage from a digital pin on the Photon board, which is why the wiring doesn't include the voltage divider.

[1] https://learn.adafruit.com/ttl-serial-camera?view=all

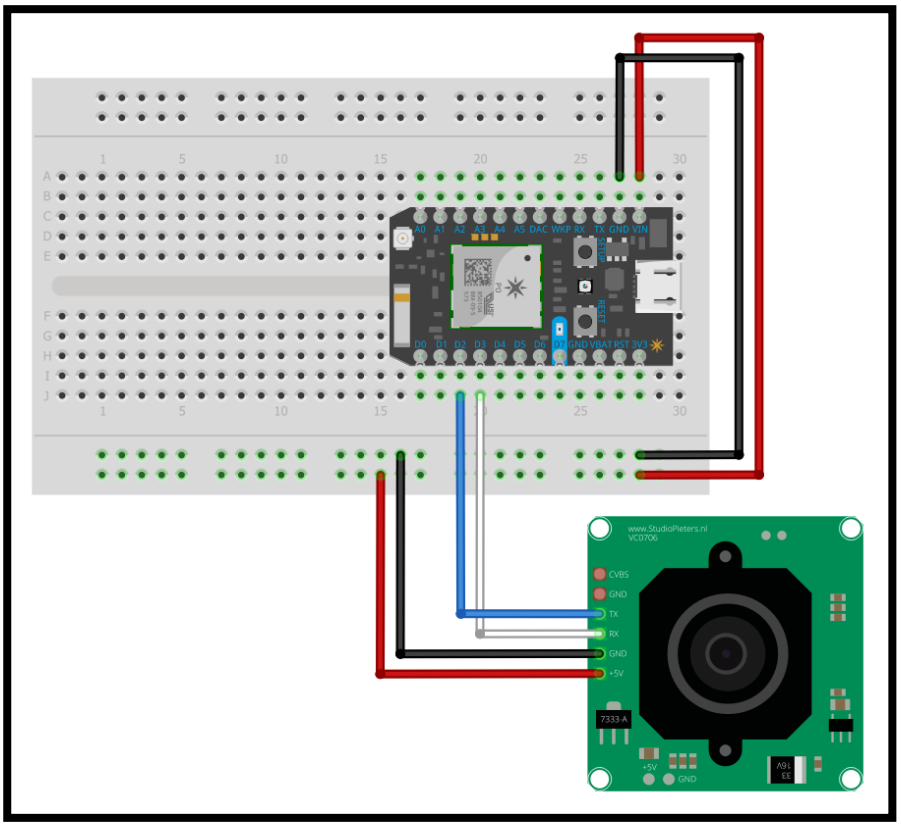

Figure 6.4.1 - Wiring of Photon board and camera

On figure 6.4.2 it can be seen how the camera is set up in software. It can be seen that we first need a serial connection between the photon board and the camera. To communicate on the serial line connected to the photon board, the camera uses two pins, which are TX and RX. TX is connected to the D2 pin and the RX is connected to the D3 pin. It can also be seen on the figure that the code includes a library called Adafruit_VC0706. This library contains a series of functions that supports the features of the camera. For instance, it contains functions to setup the camera, take snapshots, set image resolution, read the image in terms of bytes etc. The library is not initially compatible to the Photon board, since it is originally intended to work with the Arduino platform. Because of this, the library had to be adjusted by using a different library called ParticleSoftSerial for handling serial connections. In order to use and access the functions, it is required to create an instance of the Adafruit_VC0706 library. This is also seen on figure 6.4.3.

```
#include "/lib/Adafruit_VC0706/src/Adafruit_VC0706.h"
#include "ParticleSoftSerial.h"

#define RX D3
#define TX D2

//  Setup camera:
ParticleSoftSerial cameraconnection = ParticleSoftSerial(TX, RX);
Adafruit_VC0706 cam = Adafruit_VC0706(&cameraconnection);
```

Figure 6.4.2 - Setting up camera

On figure 6.4.3 It can be seen which library is used, when working with an Arduino board. It can be seen that the Arduino board uses the SoftwareSerial library instead of the ParticleSoftSerial library. Aside from this, it can be seen that the setup of the camera is very similar.

```
#include <Adafruit_VC0706.h>
#include <SoftwareSerial.h>

SoftwareSerial cameraconnection = SoftwareSerial(2, 3);
Adafruit_VC0706 cam = Adafruit_VC0706(&cameraconnection);
```

Figure 6.4.3 - Libraries and setting up camera using Arduino

The usage of the camera can be seen on figure 6.4.4. The figure indicates how the program tries to locate the camera by applying the begin function. The function sets up a baud rate for the camera and prints a message that informs about the program searching for the camera. Depending on the argument passed to the begin function, it is possible to choose a baud rate. If this is not specified explicitly through an argument, the camera uses a default baud rate of 38400. The resolution of the camera can be set with the setImageSize function, which can also be seen on figure 6.4.4. The camera is able to use three different resolutions, whereas the one used on figure 6.4.4 is the smallest one. The usage of the takePicture function can also be seen on the figure. The method results in the camera taking a picture. Lastly we print the size of the frame in bytes using the frameLength function.

```
void setup()
{
  //  Setup serial connection baud rate:
  Serial.begin(9600);
  Serial.println("Serial connection was established!");

  //  Locate camera:
  if(cam.begin()){
    Serial.println("Camera found! (Good job!)");
  }
  else{
    Serial.println("No camera found?");
    return;
  }

  //  Set resolution:
  cam.setImageSize(VC0706_160x120);
  Serial.println("Resolution was set!");

  Serial.println("Snap in 3 seconds...");
  delay(3000);

  if( !cam.takePicture()){
    Serial.println("Failed to snap!");
  }

  else{
    Serial.println("Picture taken!");
  }

  //  Print the picture bytes:
  uint32_t jpglen = cam.frameLength();
}

void loop(){}
```

Figure 6.4.4 - Locating camera and taking picture

In order to read the picture in bytes, or in other words, the camera buffer, the Adafruit_VC0706 provides a readPicture function, which can be seen on figure 6.4.5. The function returns a pointer to a uint8_t buffer, which is the camera buffer.

```
uint8_t * Adafruit_VC0706::readPicture(uint8_t n) {
  uint8_t args[] = {0x0C, 0x0, 0x0A,
                    0, 0, frameptr >> 8, frameptr & 0xFF,
                    0, 0, 0, n,
                    CAMERADELAY >> 8, CAMERADELAY & 0xFF};

  if (! runCommand(VC0706_READ_FBUF, args, sizeof(args), 5, false))
    return 0;


  // read into the buffer PACKETLEN!
  if (readResponse(n+5, CAMERADELAY) == 0)
      return 0;


  frameptr += n;

  return camerabuff;
}
```

Figure 6.4.5 - Reading the picture

A small processing program is indicated on figure 6.4.6. The purpose of the program is to listen for written bytes to a serial port on the computer.

The program contains two essential functions:

- setup()
- draw()

In the setup function it is indicated on figure 6.4.6, that the program specifically is reading on the COM3 port. This is specified through the Serial.list function, which also sets the baud rate to 9600 baud. In order to store the content written to the serial port, a file is created. The file is a text file and is called bytes.txt.

Inside the drawing function is where the actual reading of the serial port happens, as well as writing the serial port content to the bytes.txt file. The keyPressed() function, which is also seen on figure 6.4.6, is a function used to exit the program, flush and close the bytes.txt file if any key is pressed.

```
import processing.serial.*;

Serial mySerial;
PrintWriter output;
void setup() {
   mySerial = new Serial( this, Serial.list()[1], 9600 );
   output = createWriter( "bytes.txt" );
}
void draw()
{
    if (mySerial.available() > 0 ) {
         String value = mySerial.readString();
         if ( value != null ) {
              output.println( value );
         }
    }
}

void keyPressed()
{
    output.flush();
    output.close();
    exit();
}
```

Figure 6.4.6 - Processing script for reading serial port

6.5 Speakers (Mads)

As mentioned before there has to be a speaker on the inside and outside devices and they both needs to be able to play sound, but on different parameters. The sound needs to go through the digital to analog converter and through an amplifier to the speaker. There is a library available for the photon board. The library allows you to easily send the audio in a digital format. In this case the digital format is represented with unsigned 16 bit integers through the digital to analog converter. Lastly, it can be send through the amplifier and to the speaker. With this library this can be done simply by using one command to play the audio.

```
#include "speaker.h"
#include "sound.h"

const int audioFrequency = 22050; // Hz

Speaker speaker((uint16_t*)sound, sizeof(sound) / sizeof(sound[0]));
void setup() {
  speaker.begin(audioFrequency);
}
```

Figure 6.5.1 - Code for playing a sound

Figure 6.5.1 indicates an example of how this is done. First you can see the library is implemented simply by including it at the top of the file as speaker.h and sound.h where speaker.h is the library that allows you to simply connect to the DAC and the sound.h is the tone you would like to play through you speaker. Then there is a constant called audiofrequency, which is the frequency the sound is going to be played at. It can be seen that the frequency is set to be 22050 Hz. After this the speaker needs to be instantiated, this is done as shown where we have the name for the speaker in this case we call it speaker

and then we have the size of the data format that is in the sound file and the size of the array needed to be send through the digital to analog converter.
In order to play audio after the instansation, this can be done simply by calling the variable and using the dot operator and the command begin and it will send the signal through the digital to analog converter and amplifier and then speaker.

In the final code, which can be seen on figure 6.5.2, the command for the speaker to start playing is inside a function called myHandler. This function is a part of the publish subscribe function and every time the cloud tells us the other used has published the event the other photon is subscribed to it will run this function.

```
void myHandler(const char *event, const char *data)
{
  if (strcmp(event,"DoorBell")==0) {

    //play audio
    speaker.begin(audioFrequency);
    // send mail to user
    Particle.publish("SendMail");
    //stop users spotify music
    Particle.publish("musicStop",data,PRIVATE);
    // wait for 2 seconds for tune to finish
    delay(2000);
    // stop music
    speaker.end();
  }
  else{
    // do nothing
  }
}
```

Figure 6.5.2 - Doorbell event handler

On figure *6.5.3* the setup function can be seen. This function will run when an event is published in the other device to tell the photon to subscribe to this event it is set up in the set up loop where we have subscribed to an event.

```
void setup() {
  String data = String(10);
  // Here we are going to subscribe to your b
  Particle.subscribe("DoorBell", myHandler);
  // Subscribe will listen for the event budd
  // (Remember to replace buddy_unique_event_
  // myHandler() is declared later in this ap
}
```

Figure 6.5.3 - setup function

Here we subscribe to the event from the other photon and when this event is published from the other photon it will run the my handler function

DoorBell	null	Lasse-Photon-Torpedo	12/13/18 at 10:31:57 am

Figure 6.5.4 - Doorbell event
In figure 6.5.4 we can see the event being published from the other photon. and when ever the cloud tells us this event has been published we will run the my handler function

hook-sent/musicStop		particle-internal	12/13/18 at 10:54:45 am
musicStop	????	The_Doctor	12/13/18 at 10:54:45 am
SendMail	null	The_Doctor	12/13/18 at 10:54:45 am

Figure 6.5.5 - Events

On figure *6.5.5* we can see at the time when the event is published on the other photon this photon will send a mail to the user by running a IFTTT made for the purpose of sending a mail in a certain event and stop their music playing on spotify by sending a webhook to the spotify API.
while it's doing all of this it's also playing audio through the DAC to the amplifier and through the speaker to notify the user there is a person at their door.
to get more into detail on how the my handler function operates as stated earlier it runs when there is an event published by another photon to the cloud and then it will start på looking if its the right event it's trying to run on by looking at the string and if it is the right string it will start by playing audio by calling the variable instantiated earlier and using the dot operator to play the sound. while still playing audio it will publish the send mail event to the cloud to let the IFTTT applet know it should send a mail to the user and publish the music stop event to tell the webhook to stop the users music through the Spotify API.

7 Test/Verification

7.1 Mail (Lasse, Kim)

Some of the logged activities from the gmail applet can be seen on figure 7.1.1. The figure indicates that the applet is able to run, as it can be seen that the applet ran 13. december at 10:17 and 10:22. The mails that has been send to the iotmadskimlasse@gmail.com account can be seen on figure 7.1.2.

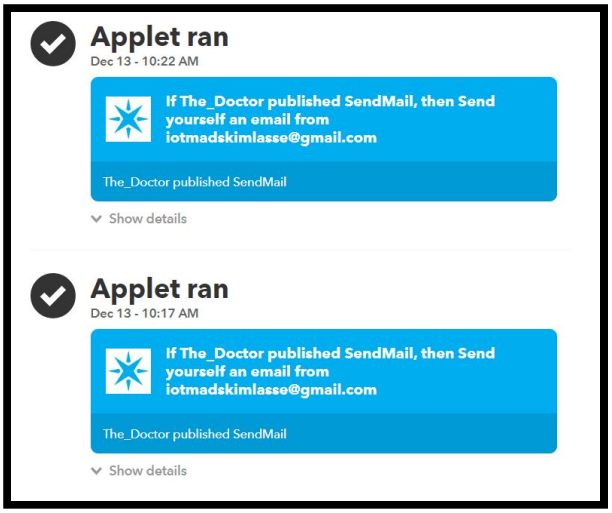

Figure 7.1.1- Logged activies from gmail applet

The inbox for the iotmadskimlasse@gmail.com account can be seen on figure 7.1.2. It can be seen that the inbox contains emails based on the IFTTT gmail applet. It can be seen that the email subject indicates the text ‘Doorbell’ and the body indicates the text ‘Somebody is at the door’, which corresponds with the aforementioned specifications regarding the gmail applet.

Figure 7.1.2 - Gmail inbox

On figure 7.1.3 the console showing that SendMail was published at 10:22 can be seen which is the same time as the mail was send and as the applet ran.

Figure 7.1.3 - Console showing send mail is published

7.2 Google Calendar (Kim)

In order to test the Google Calendar component, a single Google Calendar event was created inside calendar that belongs to the iotmadskimlasse@gmail.com account. The event was called Photon and it can be seen on figure 7.2.1. The event was set to begin at 09:40 and end at 09:50 at 5. december.

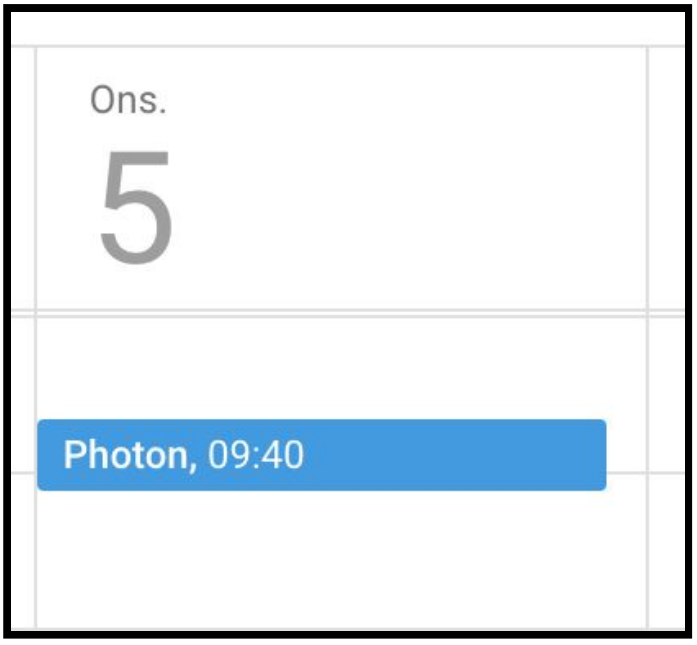

Figure 7.2.1 - Calendar Event

Figure 7.2.2 indicates a logged activities from the applets that runs if either a Google Calendar event starts or ends. It can be seen that both applets works, but the applet that runs if an event ends is a bit early. However, the applet that runs if an event starts, is able to run at the correct time. It can be seen that the applet ran at 9:40, which corresponds correctly with the Google Calendar event.

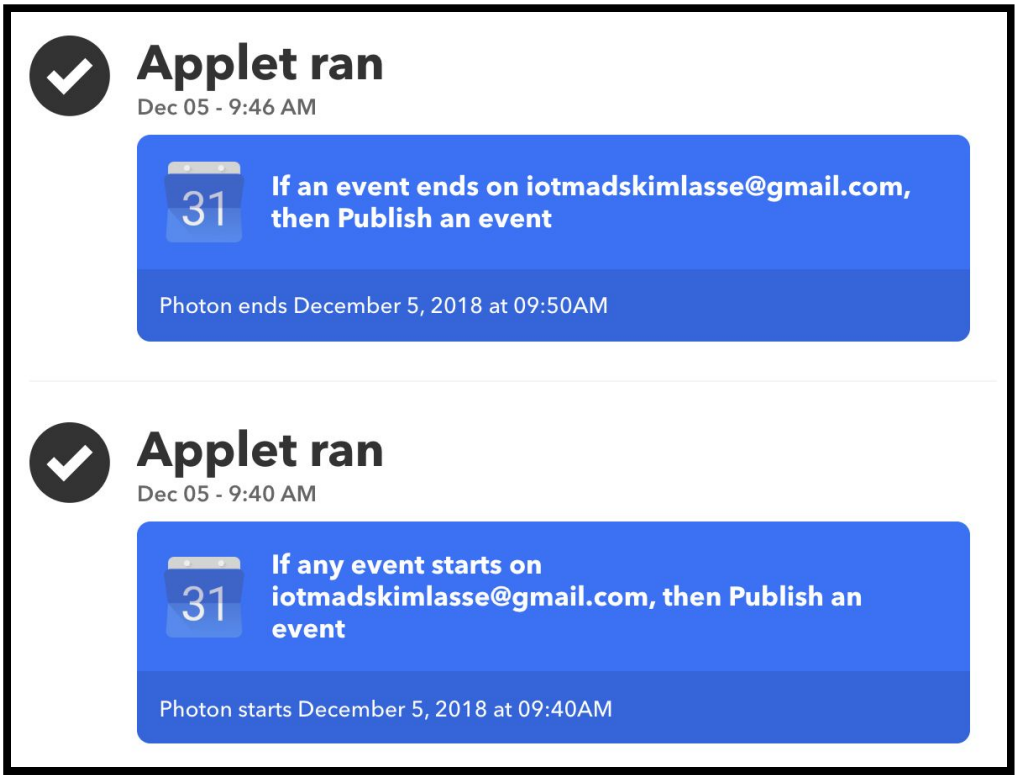

Figure 7.2.2 - Google Calendar Applet tests

7.3 Spotify (Lasse)

On figure 7.3.1 it can be seen that the event musicStop was sent by The_Doctor and then afterwards the webhook was sent from particle internal and then on the video [9] it can be seen that after the button is pressed at 10:54 the webhook is sent and the spotify player goes from being on to being paused. Some of the problems that have been with the spotify player is that the authorization token only lasts for 1 hour and has to manually be put into the integration from an external program like the spotify web api authorization example[10].

hook-sent/musicStop		particle-internal	12/13/18 at 10:54:45 am
musicStop	���●	The_Doctor	12/13/18 at 10:54:45 am
SendMail	null	The_Doctor	12/13/18 at 10:54:45 am

Figure 7.3.1 - Console

7.4 Camera (Kim)

We managed to successfully connect the Photon board and the camera a few times. During a successful connection, the output indicated on figure 7.4.1 can be seen. The output

displays a message that informs that the connection to the serial port was successful and the board is searching for the camera and lastly the camera was found.

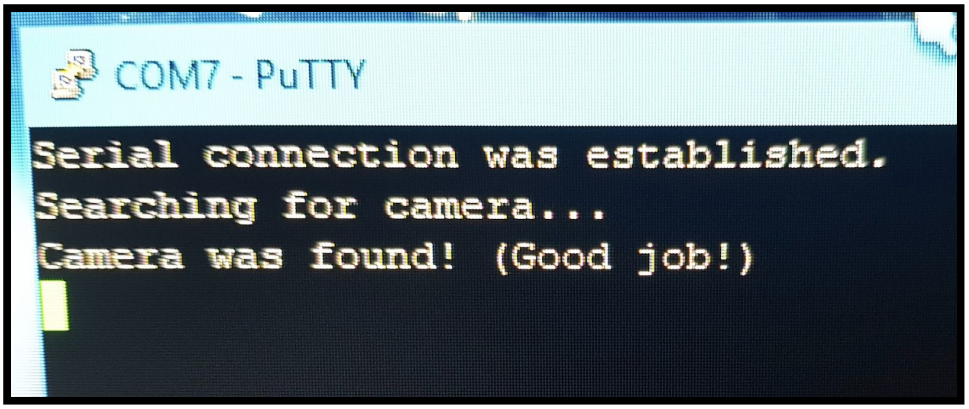

Figure 7.4.1 - Successful test of connectivity between photon board and camera

When the connection between the Photon board and the camera was unsuccessful, which happened most frequently, it displays the output indicated on figure 7.4.2. The output contains a similar message compared to the output on figure 7.4.1. The difference is that the output on figure 7.4.2 says that the camera was not found.

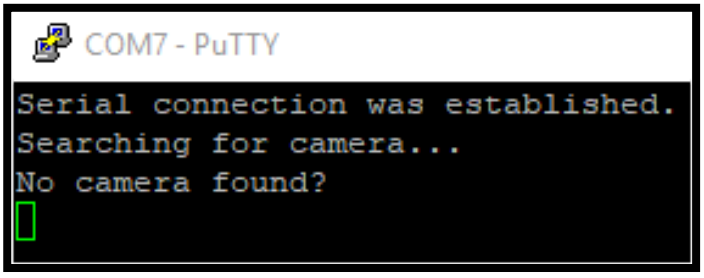

Figure 7.4.2 - Unsuccessful test of connectivity between photon board and camera

Because of the unreliable connection between the Photon board and the camera, we tried to use an Arduino Uno board instead. Using the Arduino Uno board, we observed that the camera was able to connect without any problems each time. We tried to take a snapshot with the camera and read the picture using a third party library function, which was mentioned earlier on figure 6.4.5. The results from the reading is printed to a serial port and can be seen on figure 7.4.3. It was expected to see some variations in the bytes returned from the usage of the library function, but it merely resulted in a long series of 32's. The next step would be, to read the content on the serial port and write it to a file using the Processing program. However, because of the unexpected results, it wasn't possible to decode the bytes into an image. Another important consideration to mention, is that the Arduino board does not include WiFi, unlike the Photon board, which is why is didn't make sense to change the applied development board.

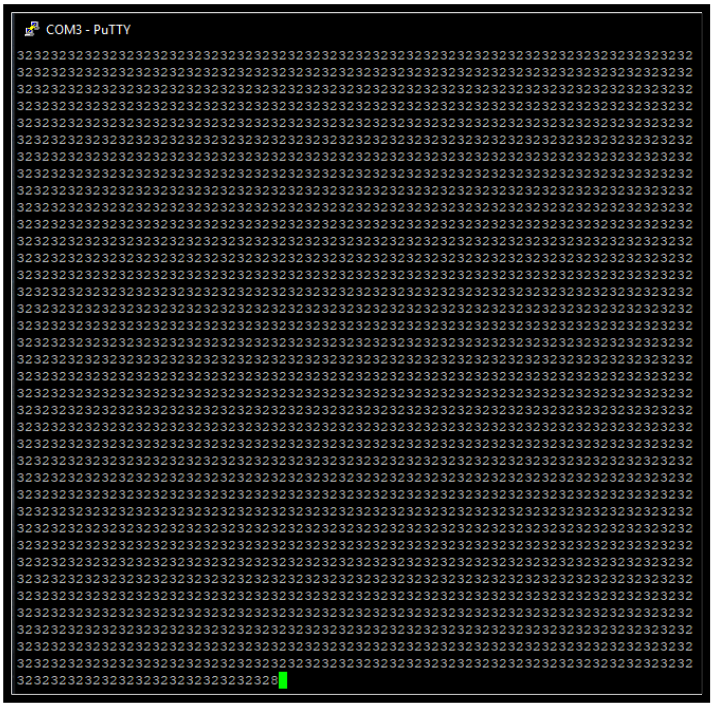

Figure 7.4.3 - Read data on serial connection using Arduino Uno

7.4.1 Measurements

Since we did not have much success connecting the camera with the Photon board, we did some measurements with the Photon board and the Arduino Uno individually connected to the camera. Based on the measurements we discovered that the supply voltage was almost similar between both boards as well as the voltage on the TX pin. However, a voltage difference of a single volt was discovered between the boards. It can be seen on figure 7.4.1.1 that the setup with the Photon board results in 3.07V at the RX pin.

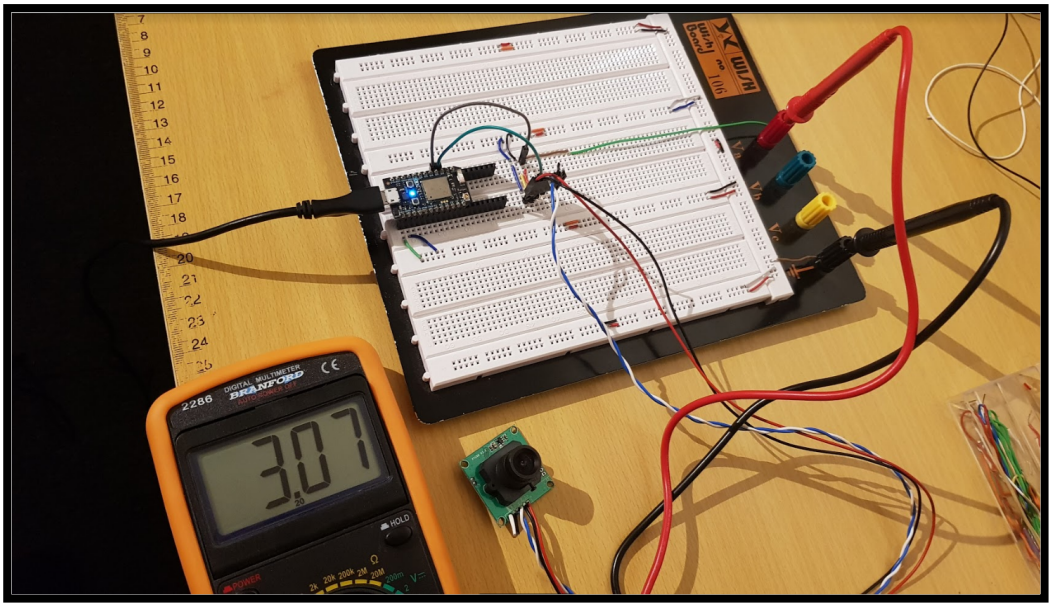

Figure 7.4.1.1 - Photon and camera measurement

It can be seen on figure 7.4.1.2 that the voltage at the RX pin was 4.17V using an Arduino Uno board. The difference might occur, since the Arduino board is able to deliver a digital output voltage up to 5V, whereas the Photon board is able to deliver up to 3.3V. In order to achieve the same digital output as the Arduino Uno, we could have used a level shifter for the Photon board. The level shifter enables shifting from one voltage to another, which could be e.g. from 3V to 5V. However, this shouldn't be necessary, since the aforementioned tutorial from Adafruit mentions that the camera should use voltage divider with a pair of 10K resistors, in order to prevent the digital output from the Arduino board to exceed 3.3V. [6] The measurement shown on figure 7.4.1.2 works with the camera, but it doesn't even include a voltage divider at the RX pin.

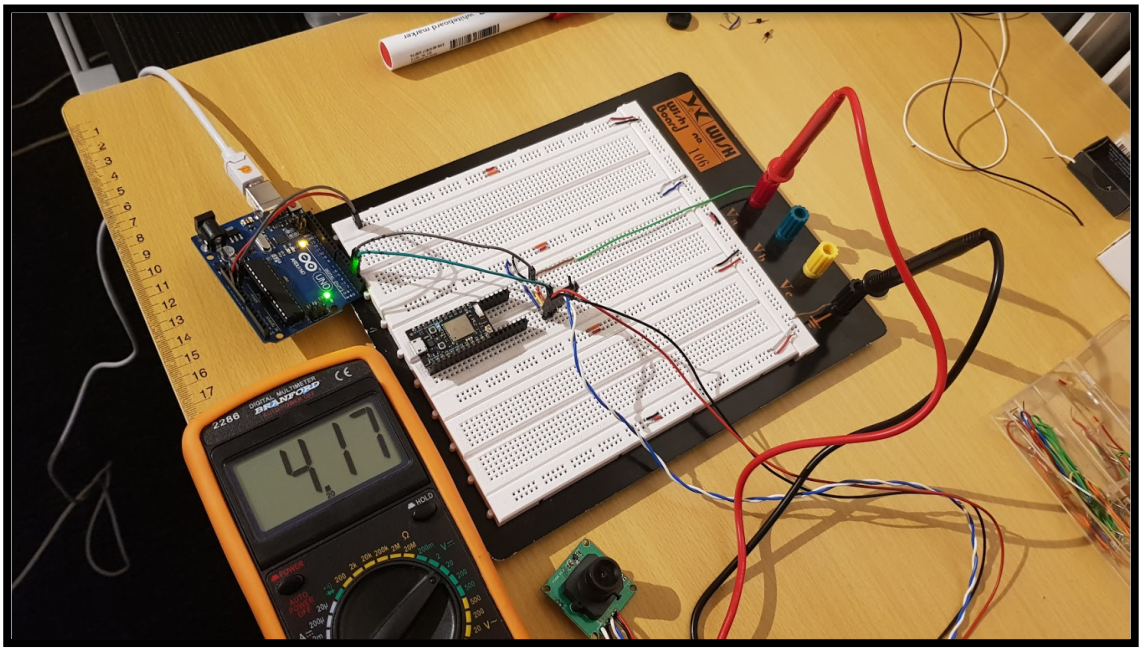

Figure 7.4.1.1 - Arduino Uno and camera measurement

Based on the results from the camera testing and verification process we did not manage to reach requirement 3.1.9 and 3.1.10. The unsuccessful experience is due to the unreliable connection from the Photon board and the camera, as well as the complexity behind reading the image from the camera and developing tools to convert the image into jpg format. The complexity behind reading the image from the camera is due to the bytes returned from the aforementioned readPicture function. The bytes didn't tell anything useful, which is fundamentally required before it makes sense to develop a converting tool.

7.5 Speaker (Mads)

To test this part of the system we have hooked everything up including both photon and connecting the speaker to a external supply as it draws to much current for the photon plugged into a USB port and it runs of 5V where the photon has a maximum voltage of 3.3V. When everything in the system is connected an hooked up the test is rather simple.
For testing you simply push the button on the other photon to publish an event to the cloud and tell the photon to play the tune through the amplifier and to verify the test you will need to listen if there is any audio. A performed test can be seen here where the button is pushed and you clearly can hear a doorbell tune playing [8]

8 Conclusion (Kim)

The developed system is able to work with a specified Google account and apply Google services in terms of Gmail and Google Calendar. These services enables the system to send an email to the specified Google account if someone presses the doorbell button and check the owner's whereabouts using Google Calendar. The system has also integrated a speaker to play a sound, when someone is pressing the doorbell button and an audio amplifier. The audio amplifier enables volume control of the playing sound from the integrated speaker. Lastly, the system has also integrated Spotify and thereby it is possible to pause currently playing music on Spotify if someone had pressed the doorbell. This feature aids the owner to know if someone is at the door, since the music might prevent the owner from hearing the doorbell sound that plays, when someone presses the doorbell button.

The system still has limitations and a few features that wasn't implemented. This will be discussed and described in the Improvements section, found under section 9.

9 Improvements (Kim)

As mentioned in the conclusion section, the system still has several limitations and a few features that wasn't implemented. The system did not include the camera feature, which was supposed to be integrated to the device outside with the doorbell. The camera was supposed to take a snapshot, when someone would press the doorbell button and store it onto Google Drive. The gmail applet could be configured to attach the latest uploaded snapshots into a mail that is to be composed and subsequently send it to the owner. However, we experienced through several tests that the connection between the camera and the Photon board was rather unreliable. We also experienced that the bytes, which represents the image was also useless and could not be converted into a jpg image. The camera feature is definitely something that could be implemented in order to further improve the system.

Another feature that was not implemented, was an additional speaker. This speaker was supposed to be integrated to the device outside with the doorbell. The speaker was supposed to play a sound if the Photon board would acknowledge that the user was not currently home using a subscription to events regarding Google Calendar. However, there is also another important aspect to consider in terms of the idea behind the speaker outside. For instance, one can assumes a scenario that involves the owner not being home and a burglar decides to rob the house. In this case, the speaker will let the burglar know about the owner not being home, and it will most likely reassure and encourage the burglar to rob the house. If the camera was already implemented it could be applied to take a snapshot of the burglar, which could be used for identification purposes and perhaps discourage the burglar from robbing the house.

10 References

1. Particle Reference Documentation: https://docs.particle.io/reference/firmware/photon/
2. Tools and features: https://docs.particle.io/guide/tools-and-features/intro/
3. Github Repository for IOT project: https://github.com/KimConcepcion/Internet-Of-Things
4. Spotify Pause: https://developer.spotify.com/console/put-pause/
5. Spotify Authorization cycle: https://developer.spotify.com/documentation/general/guides/authorization-guide/
6. Tutorial for VC0706 camera: https://learn.adafruit.com/ttl-serial-camera?view=all
7. Advice for choosing doorbell: https://www.1800doorbell.com/resources/door-chimes/best-wireless-doorbell-reviews
8. Video of test of the speaker. https://drive.google.com/file/d/1R-ShQC-IJ3qkODRU9DA1vVPlPyeKzOm9/view?usp=sharing
9. Spotify test: https://drive.google.com/open?id=1lE8E_r_BlUkAYwgtCkToIX9YwjLMgnZK
10. Code for Authorization example: https://github.com/spotify/web-api-auth-examples
11. Sleep functions for Photon: https://docs.particle.io/reference/device-os/firmware/electron/#sleep-sleep-
12. STM32F20x datasheet: https://www.st.com/resource/en/datasheet/stm32f205rb.pdf

11 Appendix

11.1 Component list

Component	Amount	Price total DKK	Link	Notes
Particle Photon Board	2			Already have
16mm Panel Mount Momentary Pushbutton	1	6,16	https://www.adafruit.com/product/1505	Button
Visaton Bærbar højtaler	1	63,36	https://dk.rs-online.com/web/p/hojttalerdrivere/7560124/	Speaker inside
SW400508-1	1	37,60	https://www.mouser.dk/ProductDetail/DB-Unlimited/SW400508-1?qs=sGAEpiMZZMve4%2fbfQkoj%252bJ205BYWtHpW2Bgb4u87R9g%3d	Speaker outside and Waterproof
PAM8403 Mini 5V Power Digital Audio Amplifier	1	29,00	https://arduinotech.dk/shop/pam8403-mini-5v-power-digital-audio-amplifier/	Audio Amplifier inside
LM386 Audio Amplifier Module.	1	21,00	https://arduinotech.dk/shop/audio-amplifier-module/	Audio Amplifier outside
TTL SERIAL JPEG CAMERA NTSC VID	1	255,28	https://www.digikey.dk/product-detail/en/adafruit-industries-llc/397/1528-1401-ND/5638296?utm_adgroup=&mkwid=s&pcrid=295222275325&pkw=&pmt=&pdv=c&gclid=Cj0KCQjwxvbdBRC0ARIsAKmec9YJa1o4cxzVdGBT39idwreLwzu3W7qIxujJSTpScUuSkt9B8AGfkmcaAgRoEALw_wcB	Camera

11.1.1 Button

This arcade button we chose because we wanted a button of a reasonable size without being to big since we are trying to make something that won't be much bigger than an ordinary doorbell. This button is ideal for our purpose since it has a good size, are easy to see when you approach the door and will not be mistaken with something else by accident.

11.1.2 Speaker inside

The Visaton bærbar højtaler we chose based on the fact that we needed a speaker with a relatively low footprint to implement on our case inside. This speakers rating of 2W max would easily be able to identify the user of the fact that there's someone at the door given our user is home.

11.1.3 Speaker outside

The SW400508-1 is a speaker with a rating of 0.5W and is waterproof which is one of the main reasons we chose this speaker since our outside system should be able to at least be somewhat water resistant because of the fact it is placed outside and we can’t guarantee that where the user places it, it wont get wet from rain or other things.

11.1.4 Amplifier inside

The inside speaker need some kind of driver so it can play a tune for when someone is using the doorbell and therefore we have chosen this amplifier for out inside speaker as it has output rating powerful to drive our speaker but a added bonus it has onboard volume control so the user can control how high they want the tune to be. This amplifier does support 2 speaker but we won’t be using the other speaker but it could be something that could be added later.