**SCHOOL OF DIGITAL MEDIA AND INFOCOMM TECHNOLOGY (DMIT)**

|  |  |
| --- | --- |
| **Date of Submission:** | Tuesday 22 August 2017 |

|  |  |
| --- | --- |
| **Prepared for:** | Ms Dora Chua |

|  |  |
| --- | --- |
| **Class:** | DBIT/FT/3B/32 |

|  |  |
| --- | --- |
| **Submitted by:** |  |

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| 1514429 | Chen Jian |
| 1514490 | Liang Xu Chao |
| 1549454 | Quah Kai Yuan |

**IOT CA2**

**Step-by-step Tutorial**

**DIPLOMA IN BUSINESS INFORMATION TECHNOLOGY**

**DIPLOMA IN INFORMATION TECHNOLOGY**

**DIPLOMA IN INFOCOMM SECURITY MANAGEMENT**

**ST0324 Internet of Things (IOT)**

**2017/2018 Semester 1**

**Table of Contents**

[Section 1 - Overview of the application 3](#_Toc491147276)

[A. What is the application about? 3](#_Toc491147277)

[B. Summary of the steps that will be described 3](#_Toc491147278)

[C. How does the final RPI set-up looks like? 5](#_Toc491147279)

[Living Room 5](#_Toc491147280)

[Baby Room 5](#_Toc491147281)

[Master Bed Room 6](#_Toc491147282)

[D. How does the application look like? 6](#_Toc491147283)

[Section 2 - Hardware requirements 10](#_Toc491147284)

[2.1 Hardware checklist 10](#_Toc491147285)

[Section 3 - Software requirement 13](#_Toc491147286)

[3.1 Software checklist 13](#_Toc491147287)

[Section 4 - Completed Fritzing Diagram (Living Room) 13](#_Toc491147288)

[4.1 Connect the components 14](#_Toc491147289)

[A. Connect LED and Resistor 14](#_Toc491147290)

[B. Connect DHT11 Sensor 15](#_Toc491147291)

[C. Connect the button 15](#_Toc491147292)

[D. Connect the I2C LCD 16](#_Toc491147293)

[E. Connect the RFID Reader 16](#_Toc491147294)

[Section 5 - Completed Fritzing Diagram (Baby Room) 17](#_Toc491147295)

[5.1 Connect the components 17](#_Toc491147296)

[A. Connect LED and Resistor 17](#_Toc491147297)

[B. Connect DHT11 Sensor 18](#_Toc491147298)

[C. Connect the button 18](#_Toc491147299)

[D. Connect PIR Motion Sensor 19](#_Toc491147300)

[E. Connect the MCP3008 19](#_Toc491147301)

[F. Add 10k ohms resistor 20](#_Toc491147302)

[G. Connect LDR 20](#_Toc491147303)

[Section 6 - Completed Fritzing Diagram (Master bed room) 21](#_Toc491147304)

[6.1 Connect the components 21](#_Toc491147305)

[A. Connect LED and Resistor 21](#_Toc491147306)

[B. Connect DHT11 Sensor 21](#_Toc491147307)

[Section 7 – Install Mosquitto 22](#_Toc491147308)

[Section 8 – AWS IoT Console 23](#_Toc491147309)

[A. Sign in to AWS IoT Console 23](#_Toc491147310)

[B. Create and Register your “Thing” 23](#_Toc491147311)

[C. Attach Policy and Thing 26](#_Toc491147312)

[Section 9 – Set up AWS DynamoDB 27](#_Toc491147313)

[**A.** **Search for AWS DynamoDB** 27](#_Toc491147314)

[Section 10 – Set up AWS S3 30](#_Toc491147315)

[**A.** **Set up for AWS S3** 30](#_Toc491147316)

[Section 11 – Create Rule 35](#_Toc491147317)

[Section 12 – Create IAM User 39](#_Toc491147318)

[Section 13 – Install the AWS Python SDK 42](#_Toc491147319)

[**A.** **Install AWS IoT SDK** 42](#_Toc491147320)

[Section 14 - Python Code – Living Room 42](#_Toc491147321)

[14.1 buttonAWS.py 42](#_Toc491147322)

[14.2 lcd.py 47](#_Toc491147323)

[14.3 records.py 49](#_Toc491147324)

[Section 15 - Python Code – Baby Room 53](#_Toc491147325)

[15.1 babyrecords.py 53](#_Toc491147326)

[15.2 buttonAWS.py 55](#_Toc491147327)

[15.3 boto\_s3.py 58](#_Toc491147328)

[Section 16 - Python Code – Master Room 61](#_Toc491147329)

[16.1 masterrecords.py 61](#_Toc491147330)

[16.2 buttonAWS.py 63](#_Toc491147331)

[Section 17 – PHP configuration 66](#_Toc491147332)

[17.1 config.php 67](#_Toc491147333)

[17.2 alldata.php 69](#_Toc491147334)

[17.3 chartdata.php 70](#_Toc491147335)

[17.4 JavaScript Libraries used 71](#_Toc491147336)

[Section 18 - Test the codes in browser 72](#_Toc491147337)

[18.1 Run buttonAWS.py 72](#_Toc491147338)

[18.2 Run records.py 72](#_Toc491147339)

[18.3 Run lcd.py 73](#_Toc491147340)

[18.4 Run boto\_s3.py 74](#_Toc491147341)

# Section 1 - Overview of the application

* 1. What is the application about?

This application is a smart home that has 3 main room. The living room, master room and the baby room. The application helps the user to monitor their house and control sensors remotely from their smart devices.

For the living room, the user is able to use their UUID card to enter/leave the house. Besides that, there are CCTV which is being simulated by taking photo every 5 minutes. The room also includes temperature and humidity sensor that allow user to see their room temperature and humidity. It also includes 3 LED that allow user to control manually or remotely through the smart home dashboard from their smart device.

In the Baby room, there are temperature and humidity sensor to capture the temperature and humidity of the room. Light sensors to record light value and LED that can be controlled remotely or manually. Besides that, there are also motion sensor that used to capture huge movement of the baby. Once movement is being detected, an image will be captured. This is to prevent baby from falling from bed. As parents will be notified from email and react quickly if anything happens to the baby.

In the master bed room. It includes temperature sensor and light sensor that allow user to see their room temperature on the web screen. It also includes controlling the led by pressing the button.

The data collected will all be stored in AWS S3 and Dynamo Database.

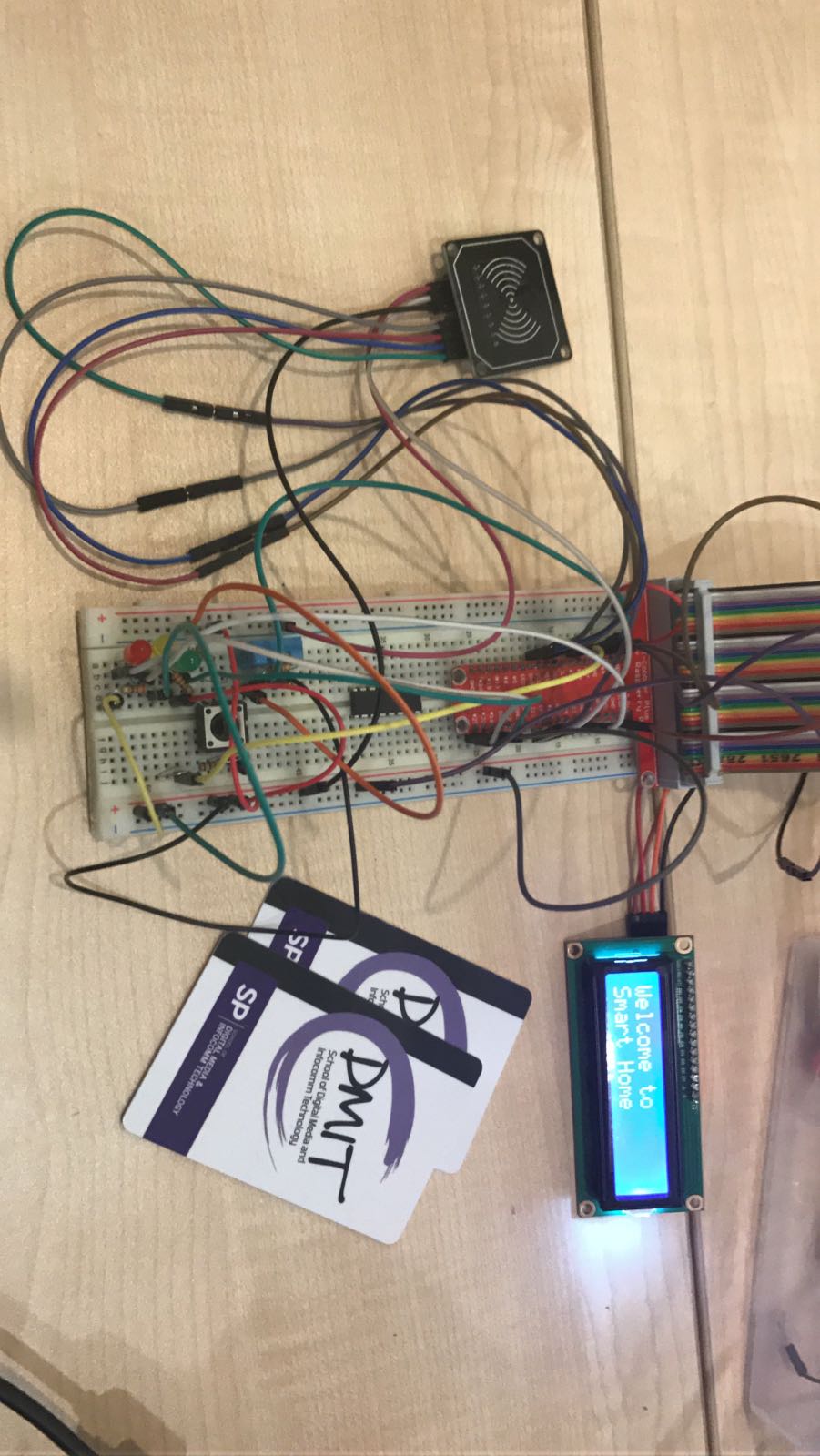
* 1. Summary of the steps that will be described

Provide a bullet list of the steps that will be covered in the other parts of this tutorial

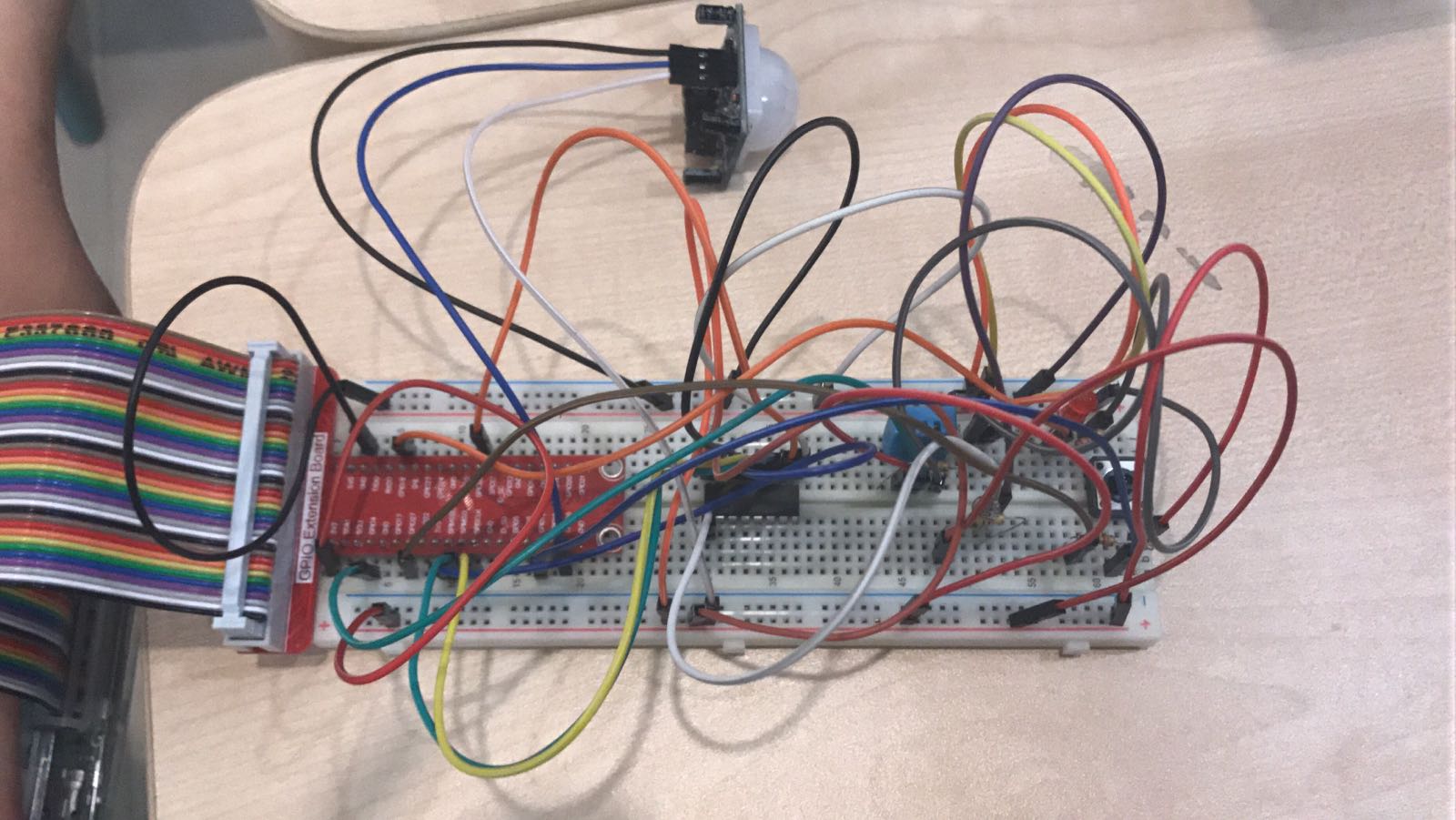
|  |  |  |
| --- | --- | --- |
|  | Section | Description |
|  | Overview | Get an overview of what you will do for the lab |
|  | Hardware Check list | Check what are the hardware that we need to build the application and gain an understanding why we need to use them   * Raspberry Pi * MCP3008 Analog-Digital Converter * Light Resistant Diode (LDR) * 10K Ω resistor * PIR Motion Sensor * LED * 330K Ω resistor * I2C LCD * RFID Reader * Button * DHT11 Sensor * Pi Camera |
| 3) | Software Check list | Check what are the software that we need to build the application |
| Sections 4 to 18 provides the step-by-step instructions to set up the application | | |
| 4) | Completed Fritzing Diagram for Living Room | Completed Fritzing Diagram that shows how completed circuit looks like with step by step description for the connection for the Living Room |
| 5) | Completed Fritzing Diagram for Baby Room | Completed Fritzing Diagram that shows how completed circuit looks like with step by step description for the connection for the Baby Room |
| 6) | Completed Fritzing Diagram for Master Room | Completed Fritzing Diagram that shows how completed circuit looks like with step by step description for the connection for the Master Room |
| 7) | Install Mosquitto | Step by step description on how to install Mosquitto |
| 8) | Sign into AWS Console | Step by step description on how to connect your Pi to AWS |
| 9) | Set up AWS DynamoDB | Step by step description on how to set up the AWS DynamoDB |
| 10) | Set up AWS S3 | Step by step description on how to set up the AWS S3 |
| 11) | Create Rule | Step by step description on how to create rule for different tables |
| 12) | Create IAM | Step by step description on how to create IAM user |
| 13) | Install the AWS Python library | Install AWS library to your Pi |
| 14) | Python Code - Living Room | Step By Step description on how to set up the Living Room python script |
| 15) | Python Code - Baby Room | Step By Step description on how to set up the Baby Room python script |
| 16) | Python Code - Master Room | Step By Step description on how to set up the Master Room python script |
| 17) | PHP configuration | Step By Step description on how to set up the PHP configuration |
| 18) | Test of the programme | Test the python scripts and ensure they receive the expected results |

* 1. How does the final RPI set-up looks like?

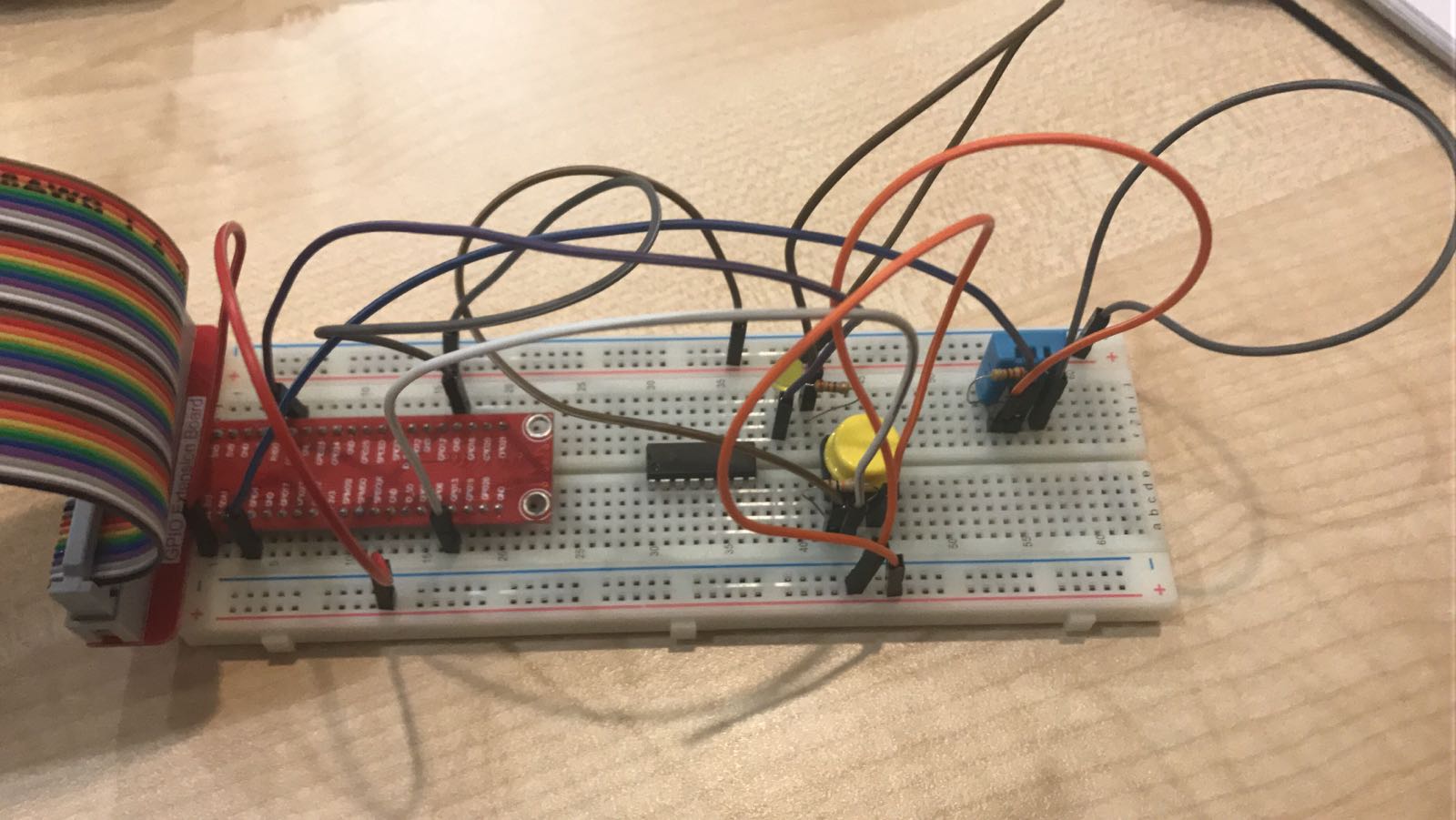
### Living Room



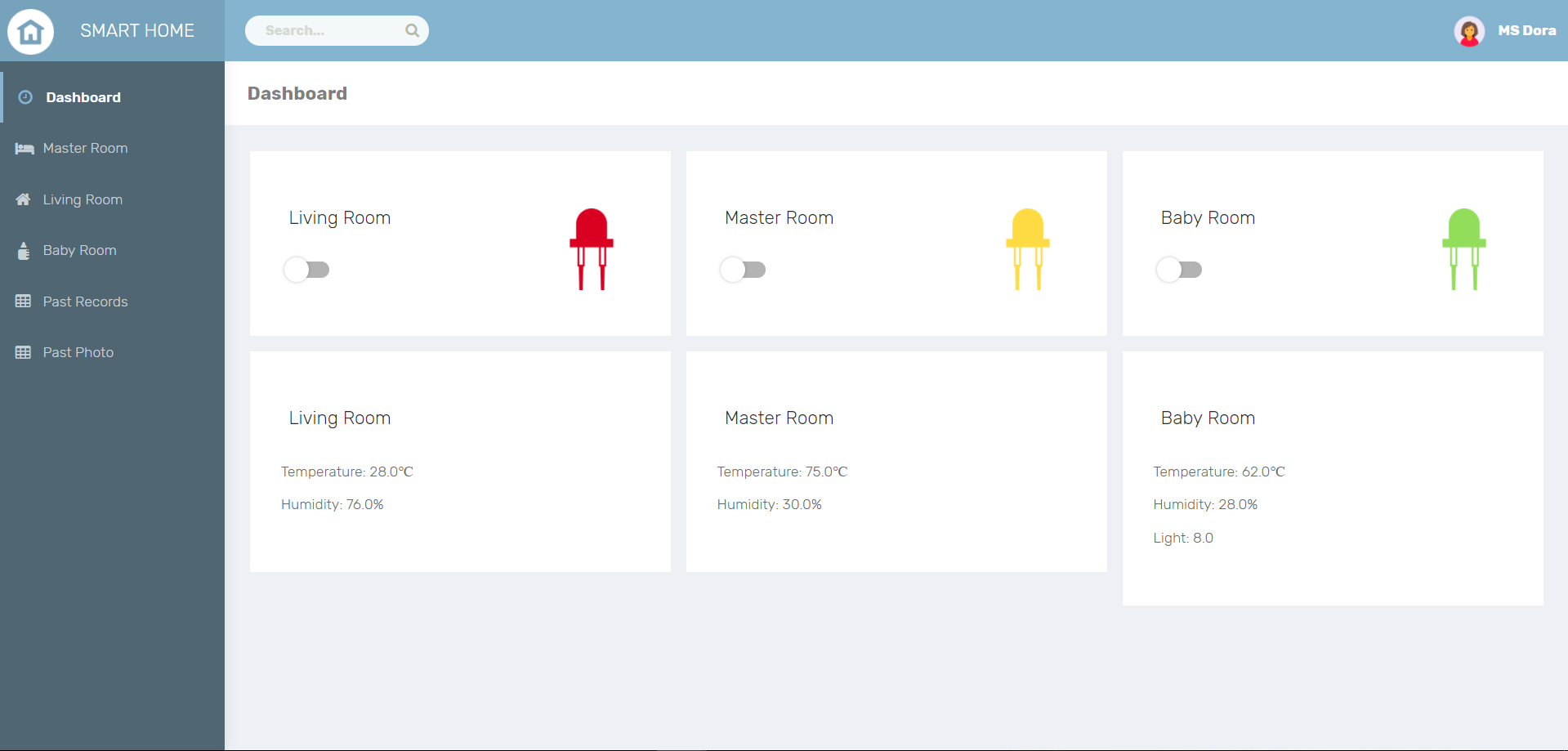
### Baby Room



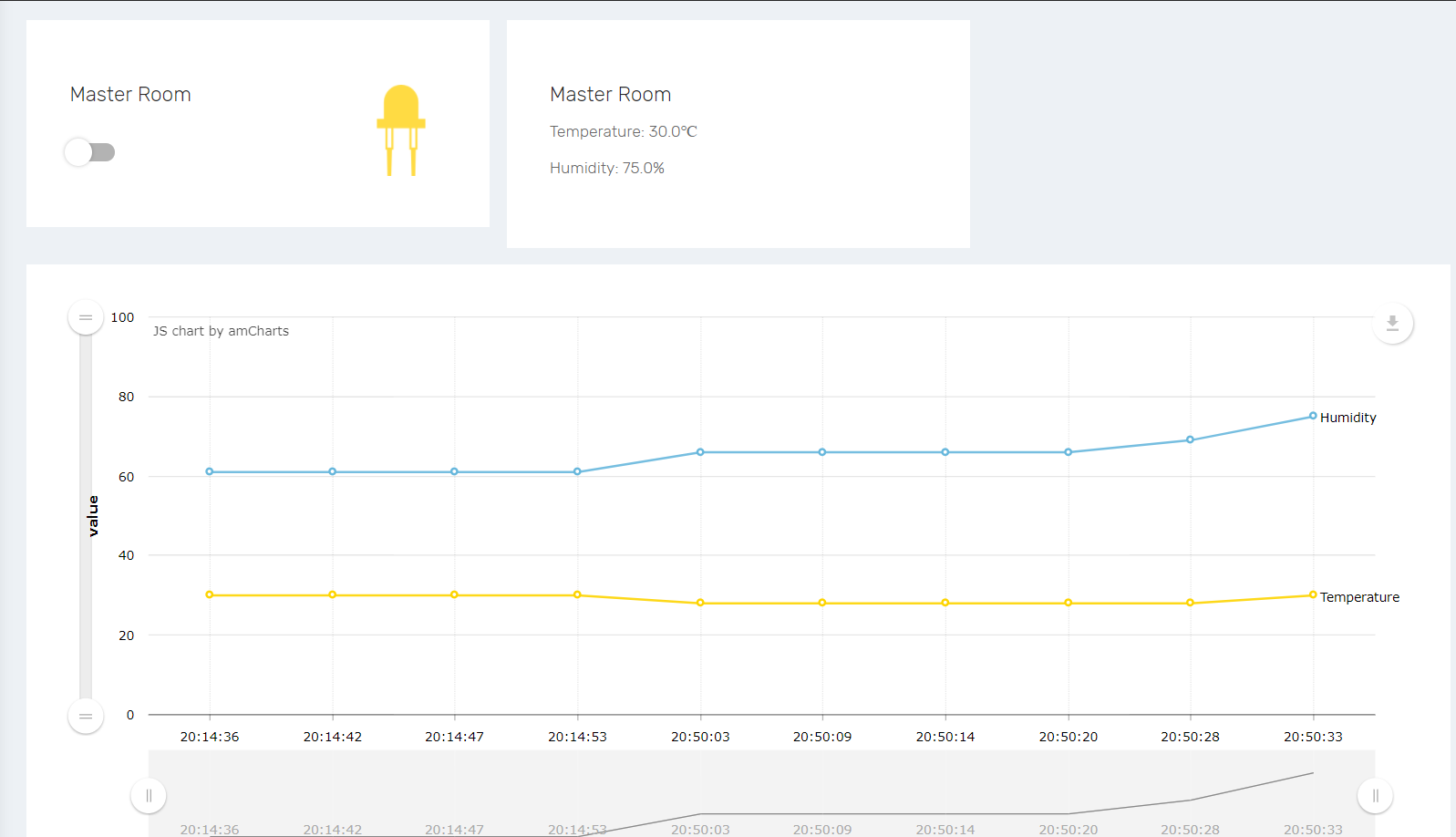
### Master Bed Room



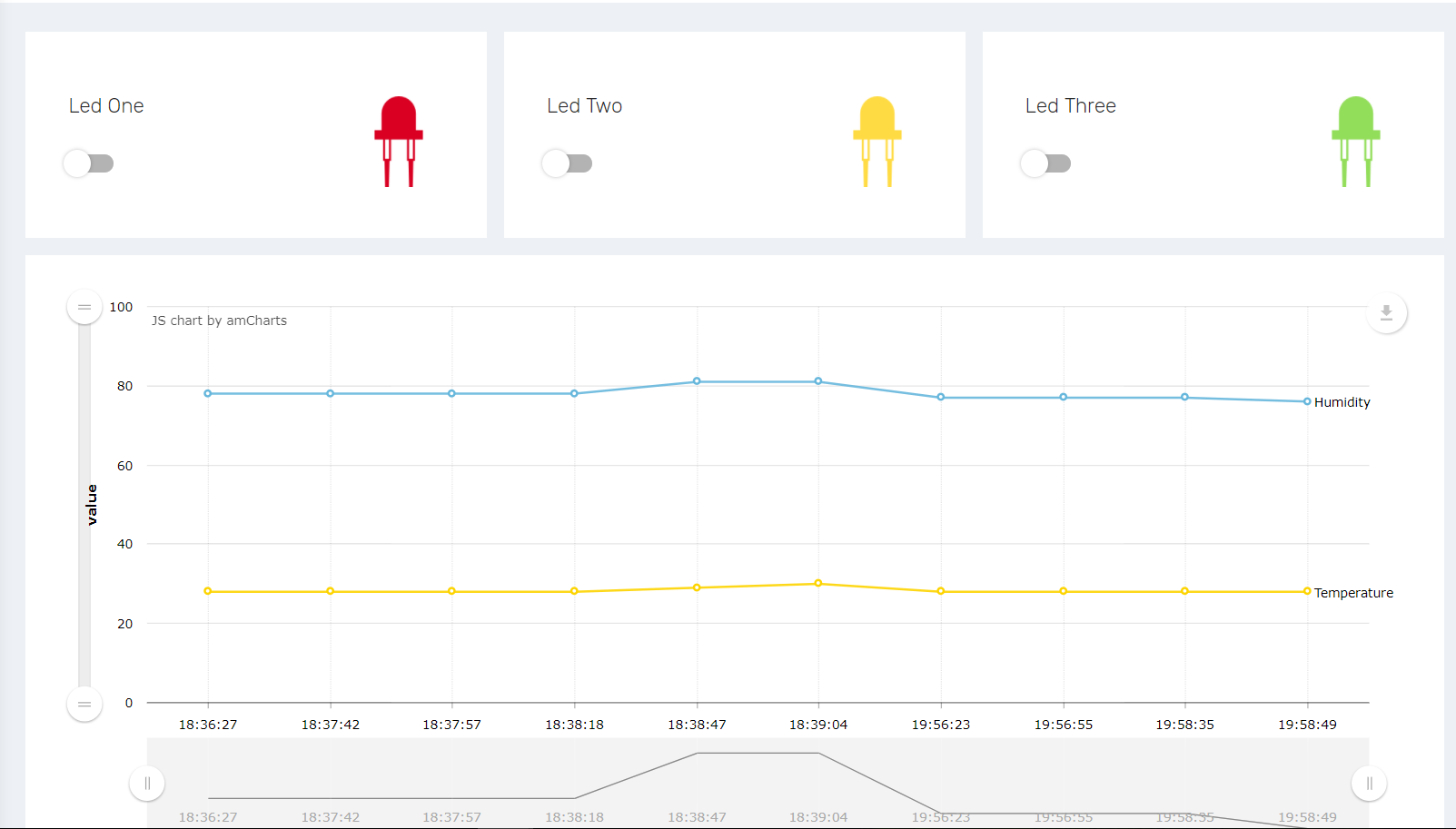
* 1. How does the application look like?



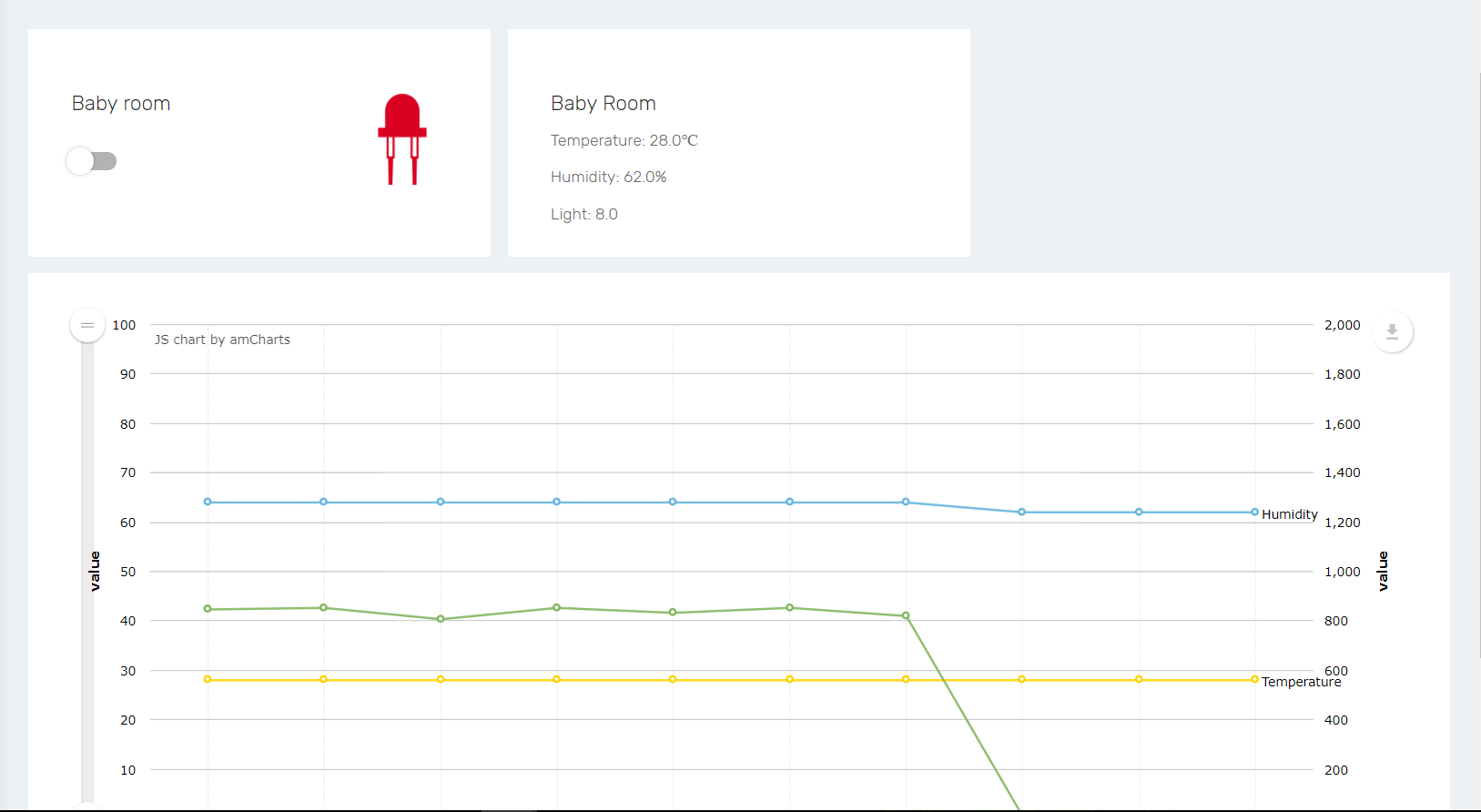
Index.php



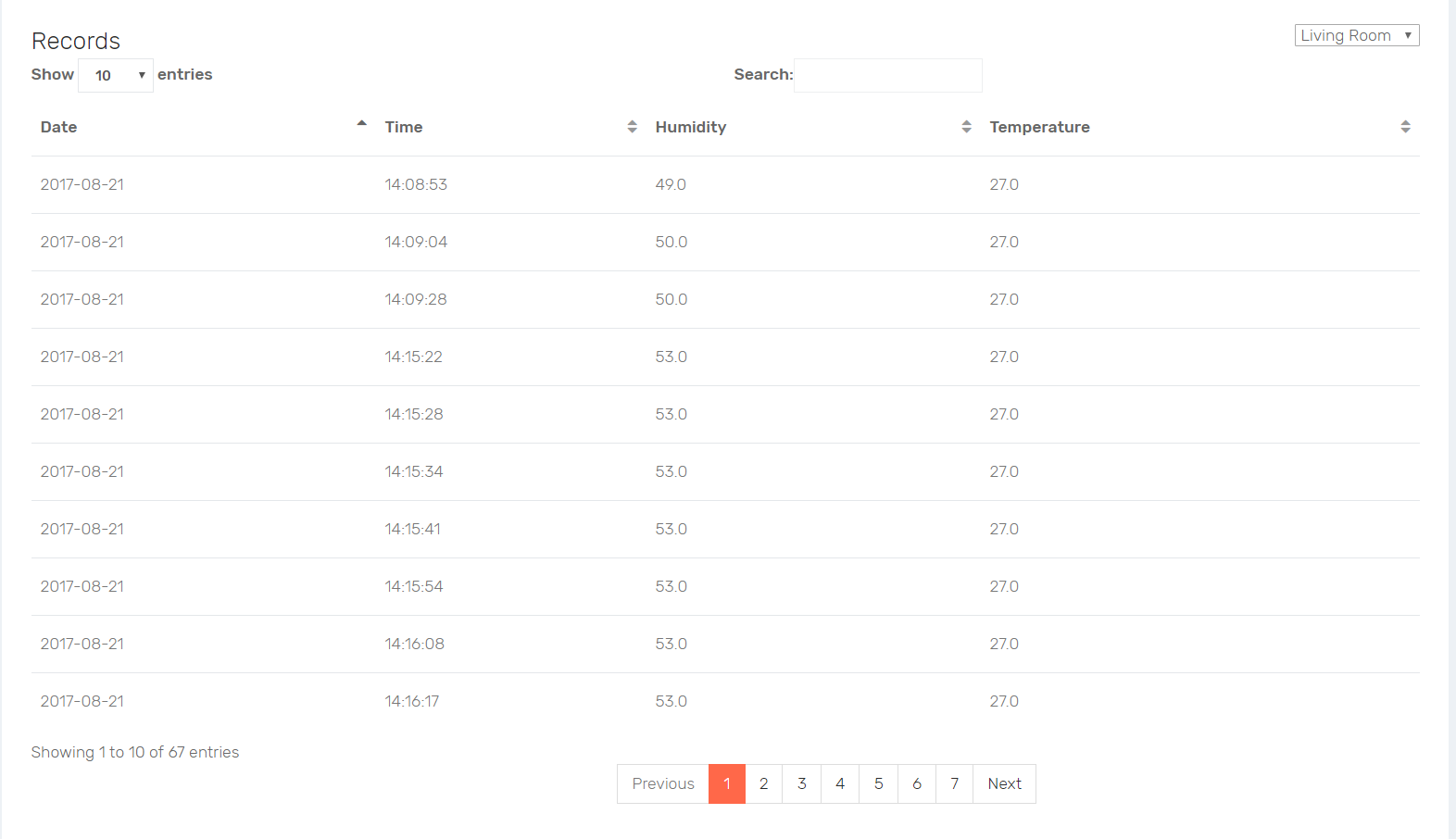
MasterRoom.php



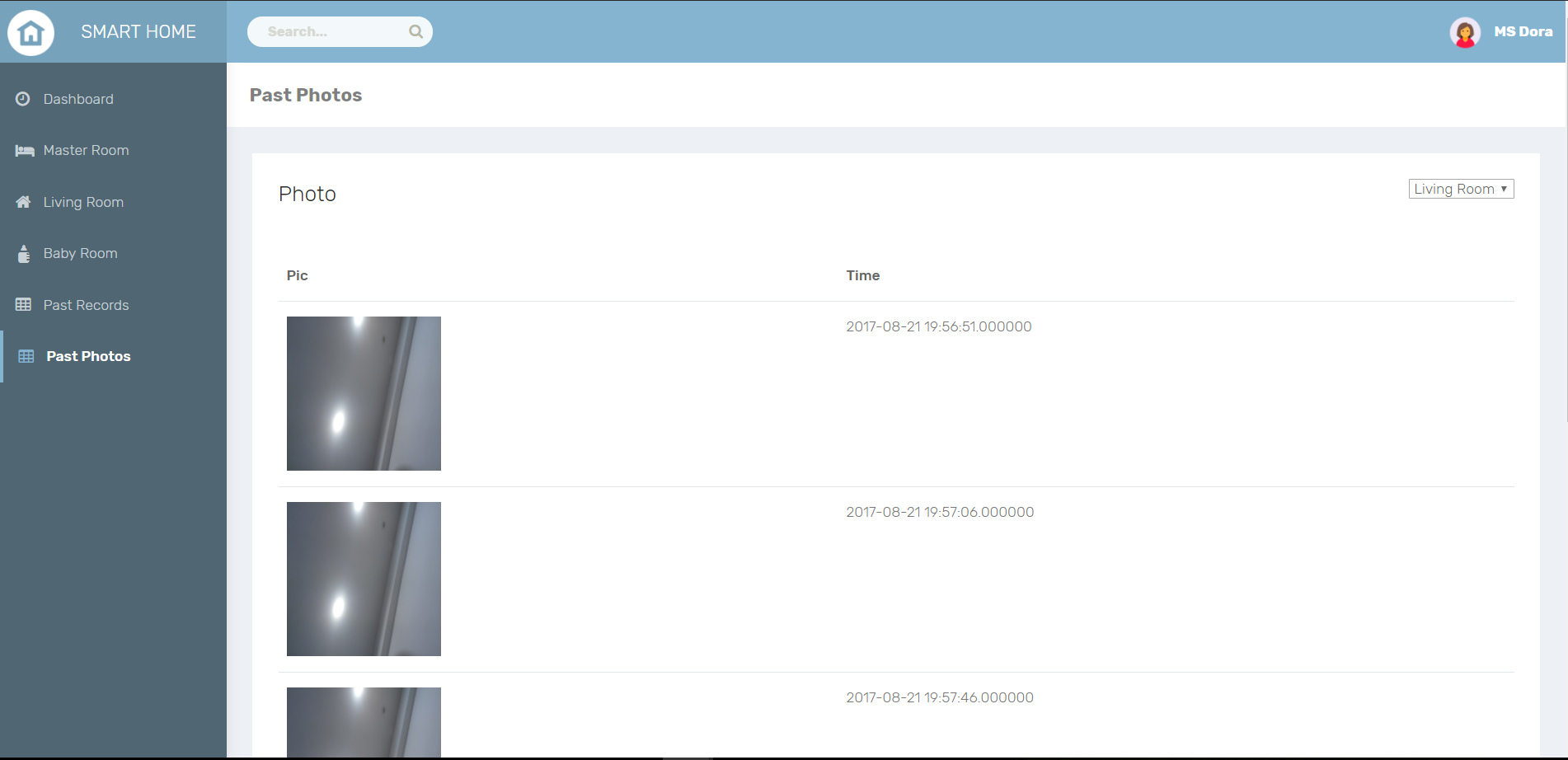
LivingRoom.php



BabyRoom.php



History.php



Picture.php

# Section 2 - Hardware requirements

2.1 Hardware checklist

|  |
| --- |
| 1. A PIR Sensor can tell when something nearby moves.   The sensor detects the pattern of infrared energy in its surroundings. If the pattern changes, the sensor outputs a high signal.  The sensor has 3 pins.   * + VCC – Connect this to power (5V or 3.3V)   + VOUT – Connect this to a GPIO pin to read its value   + GND – Connect this to ground |
| 1. A light-emitting diode (LED) that emits light when activated.  * The longer leg (anode) should be connected to the positive supply of the circuit (power) * The shorter leg (cathode) should be connected to the negative side of the power supply (ground)   • LEDs will only work if power is supplied the correct way round • You will not break the LEDs if you connect them the wrong way – they just don’t light. • If the LED doesn’t light when you finish your lab later, maybe you have connected it the wrong way |
| 1. A 330Ω resistor that used to reduce the current flow and to prevent burning the pi.      * The colours of a 330Ω resistor are Orange, Orange, Black, Black, and Brown. |
| 1. • Light-Resistant Diodes (LDR) are light sensitive resistors which change resistance based on how much light they are exposed to. • The more light a LDR receives, the less resistant it becomes, i.e. let’s more current flow • When it’s in the dark, the resistance is very high |
| 1. A MCP3008 that is great option for reading simple analog signal such as light sensor. |
| 1. A 10k Ω resistor that used to reduce the current flow and to prevent burning the pi.    * The colours of a 10k Ω resistor are Brown, Black, Orange and Gold. |
|   **BUTTON**   1. The picture here shows a tactile push-button    * Note that it has 4 'legs'. These legs are the ones that send the signals.    * Note that unlike a LED which is an actuator (output), push-buttons act as sensors (inputs)    * To know whether a push-button is pressed or released, we can detect its HIGH or LOW state, which are passed through the 'legs' |
| 1. The DHT11 is a basic, ultra-low-cost digital temperature and humidity sensor.   **DHT11**   * + - It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed).     - It’s fairly s imple to use, but requires careful timing to grab data.     - The only real downside of this sensor is you can only get new data from it once every 2 seconds     - The sensor that has been given to you has 4 pins. o VCC – this is the one you connect to power * DATA – this is an output value you read to determine the temperature * NC – Stands for no connection * GND – this is the one you connect to ground |
| 1. LED screen  * Adding a LCD display in your IoT projects will definitely * Spice it up by one notch! LCD displays are a fun and easy * Way to have your microcontroller project talk back to you.   **i2c LCD Screen**   * + LCDs are available in tons of colours and sizes. For example, you might have 8x1 LCDs or 20x4 LCDs   + For this lab, we will use the commonly available 16x2 LCD * Which can display up to 32 characters.   + We will use the i2C version which require you to connect * Only 2 GPIO pins to your Raspberry Pi.   + If you are buying your own LCDs, do make sure you buy the i2C version though they might cost a bit more. The non‐i2C versions require you to connect double the number of GPIO pins! |
| 1. NFC   **RFID/ NFC reader**  **MFRC522**   * RFID technology is commonly used in many applications to “tag” items for the purpose of identification or tracking. * For example, an RFID tag attached to an automobile during production can be used to track its progress through the assembly line; RFID tagged pharmaceuticals can be tracked through warehouses; and implanting RFID microchips in livestock and pets allows animals to be identified. * In this lab, you will write a simple Python program that is able to read RFID cards/ tags with a MFRC522 card reader module attached to RPi. * This reader is also capable of reading NFC Mifare a cards. * NFC is based on RFID protocols but goes farther than RFID as it is capable of two‐way communication—unlike RFID’s one‐directional |

# Section 3 - Software requirement

3.1 Software checklist

The software that needs to be installed on the Raspberry Pi for this lab would be the following.

* + 1. XAMPP act as the server
    2. NetBeans for coding
    3. AWS DynamoDB to store data
    4. AWS S3 to store images
    5. AWS IoT

# Section 4 - Completed Fritzing Diagram (Living Room)

Configure the hardware on your Raspberry Pi to the Fritzing diagram shown and follow the steps for the connection.

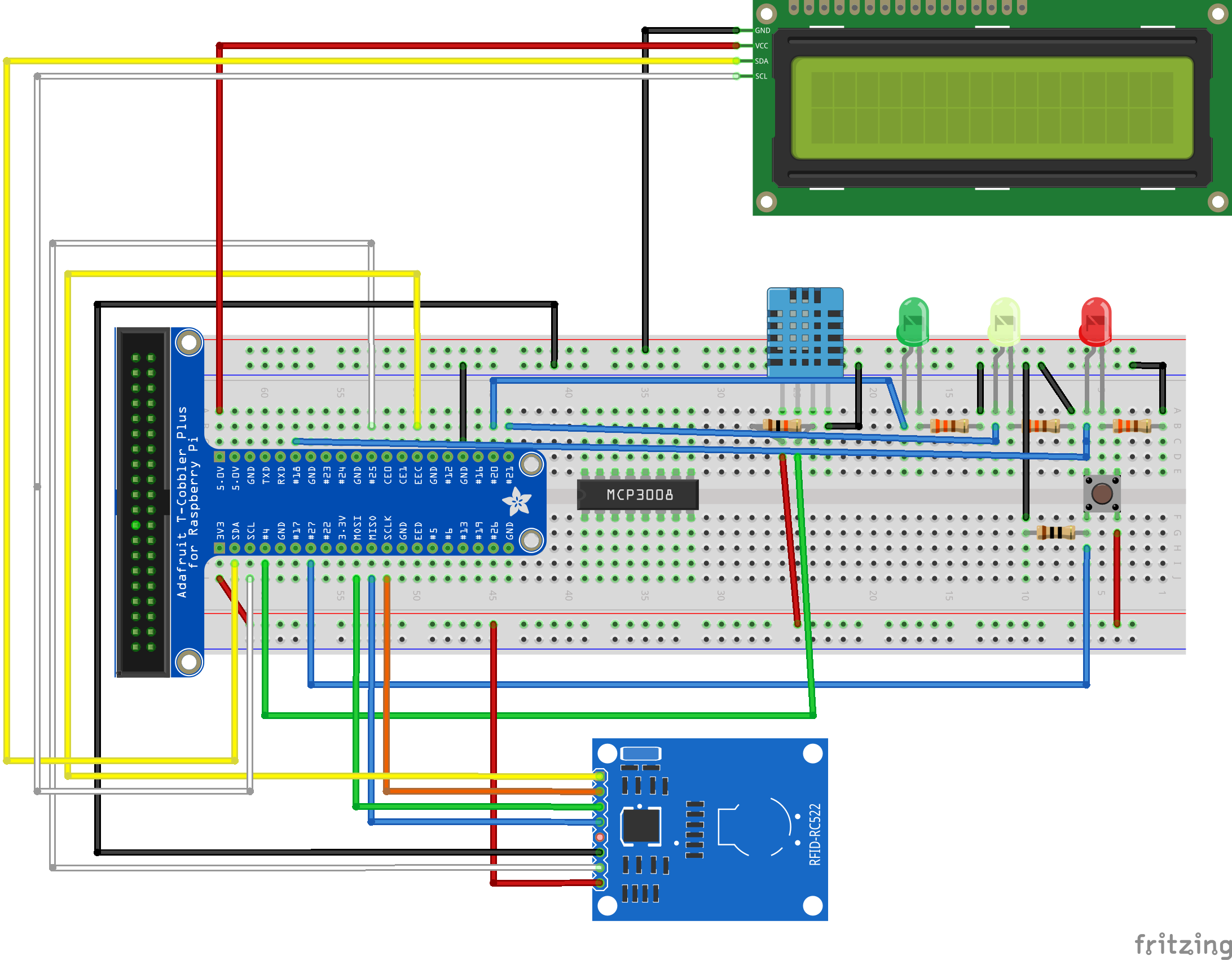


Figure : Fritzing Diagram - living room

## 4.1 Connect the components

|  |  |
| --- | --- |
| No. | Task |
| Connect LED and Resistor | |
| a) | Plug in the Red LED into the breadboard as shown. Place the longer end of the LED nearer to the T-Cobbler Kit. |
| b) | Plug in the 330 Ω resistor into the breadboard as shown. One end of the resistor should be in the same column as the short leg of the LED. |
| c) | Using a black cable, connect it one end to GND and another end to top positive (+) power rail |
| d) | **LED**  (i) Using a **blue** cable, connect the long end LED to **BCM18** |
| e) | **Resistor (Orange-Orange-Brown)**  (i) Using a **black** cable, connect the free end of the 330 Ω resistor to **Top positive(+) power rail** |
| f) | Plug in the Yellow LED into the breadboard as shown. Place the longer end of the LED nearer to the T-Cobbler Kit. |
| g) | Plug in the 330 Ω resistor into the breadboard as shown. One end of the resistor should be in the same column as the short leg of the LED. |
| h) | **LED**  (i) Using a blue cable, connect the long end LED to **BCM21** |
| i) | **Resistor (Orange-Orange-Brown)**  (i) Using a black cable, connect the free end of the 330 Ω resistor to **Top positive(+) power rail** |
| j) | Plug in the Green LED into the breadboard as shown. Place the longer end of the LED nearer to the T-Cobbler Kit. |
| k) | Plug in the 330 Ω resistor into the breadboard as shown. One end of the resistor should be in the same column as the short leg of the LED. |
| l) | **LED**  (i) Using a blue cable, connect the long end LED to **BCM20** |
| m) | **Resistor (Orange-Orange-Brown)**  (i) Using a black cable, connect the free end of the 330 Ω resistor to **Top positive(+) power rail** |
| Connect DHT11 Sensor | |
| a) | Place the DHT11 Sensor on the breadboard as shown in figure 1. |
| b) | Plug in the 10 Ω resistor into the breadboard as shown in figure 1. |
| c) | Using a red cable, connect it one end to 3.3V and another end to bottom positive (+) power rail |
| d) | **DHT11 Pin 1**  (i) Using a red cable, connect it to **bottom positive (+) power rail** |
| e) | **DHT11 Pin 2**  (i) Using a green cable, connect it to **BCM4** |
| g) | **DHT11 Pin 4**  (i) Using a black cable, connect it to **Top positive (+) power rail** |
| Connect the button | |
| a) | Place the button on the breadboard as shown in figure 1. |
| b) | Plug in the **10K Ω** resistor into the breadboard as shown. One end of the resistor should be in the same column as the left pin of the button. |
| c) | **Button**  (i) Using a **blue** cable, connect the left pin of the push button to **BCM27**  (ii) Using a **red** cable, connect the right pin of the push button to **bottom positive (+) power rail** |
| d) | **10K** Ω **Resistor (Brown-Black-Orange)**  (i) Using a **black** cable, connect the free end of the 10K Ω resistor to **Top positive (+) power rail** |
| D. Connect the I2C LCD | |
| a) | Connect I2C LCD as following:   |  |  |  | | --- | --- | --- | | Jumper Colour | I2C LCD Pin | RPI Pin | | Black | GND | Top positive(+) power rail | | Red | VCC | 5.0V | | Yellow | SDA | SDA | | White | SCL | SCl | |
| Connect the RFID Reader | |
| a) | There are about 9 pins on the MFRCF522 Reader.  We will be only using 7 pins  Connect the pins as following:   |  |  |  | | --- | --- | --- | | Jumper Color | I2C LCD Pin | RPI Pin | | Yellow | SDA | CE0 | | Orange | SCK | SCLK | | Green | MOSI | MOSI | | Blue | MISO | MISO | | Black | GND | Top positive(+) power rail | | White | RST | GPIO25 | | Red | 3.3V | Bottom positive(+) power rail | |

# Section 5 - Completed Fritzing Diagram (Baby Room)

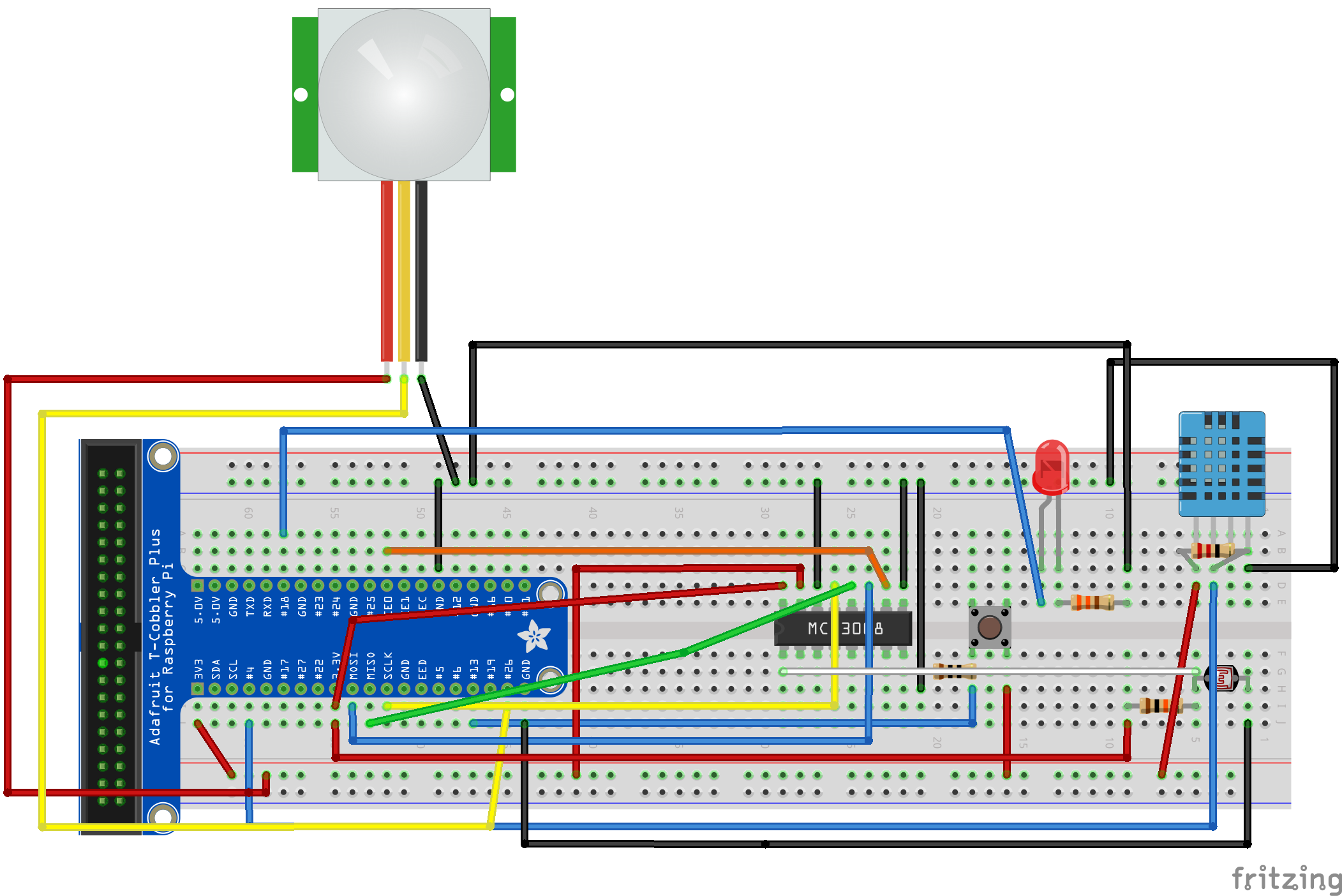


Figure : Fritzing diagram - baby room

## 5.1 Connect the components

|  |  |
| --- | --- |
| No. | Task |
| Connect LED and Resistor | |
| a) | Plug in the Red LED into the breadboard as shown. Place the longer end of the LED nearer to the T-Cobbler Kit. |
| b) | Plug in the 330 Ω resistor into the breadboard as shown. One end of the resistor should be in the same column as the short leg of the LED. |
| c) | Using a black cable, connect it one end to GND and another end to top positive (+) power rail |
| d) | **LED**  (i) Using a **blue** cable, connect the long end LED to **BCM18** |
| e) | **Resistor (Orange-Orange-Brown)**  (i) Using a **black** cable, connect the free end of the 330 Ω resistor to **Top positive(+) power rail** |
| Connect DHT11 Sensor | |
| a) | Place the DHT11 Sensor on the breadboard as shown in figure 1. |
| b) | Plug in the 10 Ω resistor into the breadboard as shown in figure 1. |
| c) | Using a red cable, connect it one end to 3.3V and another end to bottom positive (+) power rail |
| d) | **DHT11 Pin 1**  (i) Using a red cable, connect it to **bottom positive (+) power rail** |
| e) | **DHT11 Pin 2**  (i) Using a green cable, connect it to **BCM4** |
| g) | **DHT11 Pin 4**  (i) Using a black cable, connect it to **Top positive (+) power rail** |
| Connect the button | |
| a) | Place the button on the breadboard as shown in figure 1. |
| b) | Plug in the **10K Ω** resistor into the breadboard as shown. One end of the resistor should be in the same column as the left pin of the button. |
| c) | **Button**  (i) Using a **blue** cable, connect the left pin of the push button to **BCM13**  (ii) Using a **red** cable, connect the right pin of the push button to **bottom positive (+) power rail** |
| d) | **10K** Ω **Resistor (Brown-Black-Orange)**  (i) Using a **black** cable, connect the free end of the 10K Ω resistor to **Top positive (+) power rail** |
| Connect PIR Motion Sensor | |
| a) | Connect PIR Motion Sensor as following:   |  |  |  | | --- | --- | --- | | Jumper Colour | PIR Pin | RPI Pin | | Red | VCC | Top positive(+) power rail | | Black | GND | Bottom positive(+) power | | Yellow | VOUT | BCM26 | |
| Connect the MCP3008 | |
| a) | Insert the MCP3008 chip in the middle of the breadboard, with the small notch facing the T-Cobbler kit. |
| b) | The MCP3008 chip has a total of 16 pins, prepare 9 jumper cable as you will being using 9 of the pins. |
| c) | **Pin 16 – VDD**  (i) Using a red cable, connect it to 3V3 |
| d) | **Pin 15 – VREF**  (i) Using a red cable, connect it to 3V3 |
| e) | **Pin 14 – AGND**  (i) Using a black cable, connect it to GND |
| f) | **Pin 13 – CLK**  (i) Using a yellow cable, connect it to BCM11 - SCLK |
| g) | **Pin 12 – DOUT**  (i) Using a green cable, connect it to BCM9 - MISO |
| h) | **Pin 11 – DIN**  (i) Using a blue cable, connect it to BCM10 - MOSI |
| i) | **Pin 10 – CS/SHDN**  (i) Using a orange cable, connect it to BCM8 – CE0 |
| j) | **Pin 9 – DGND**  (i) Using a black cable, connect it to GND |
| k) | **Pin 1**  (i) Using a white cable, connect it to LDR |
| Add 10k ohms resistor | |
| a) | Add a 10k ohms resistor to the breadboard   * One end of the resistor should connected to 3.3V * The other end of the resistor should connected to Pin 1 of the MCP3008 ADC and the LDR. |
| Connect LDR | |
| a) | Add a LDR to the breadboard   * One end of the LDR should connected to GND * The other end of the LDR should connected to Pin 1 of the MCP3008 ADC and the 10k ohms resistor |
|  |  |

# Section 6 - Completed Fritzing Diagram (Master bed room)

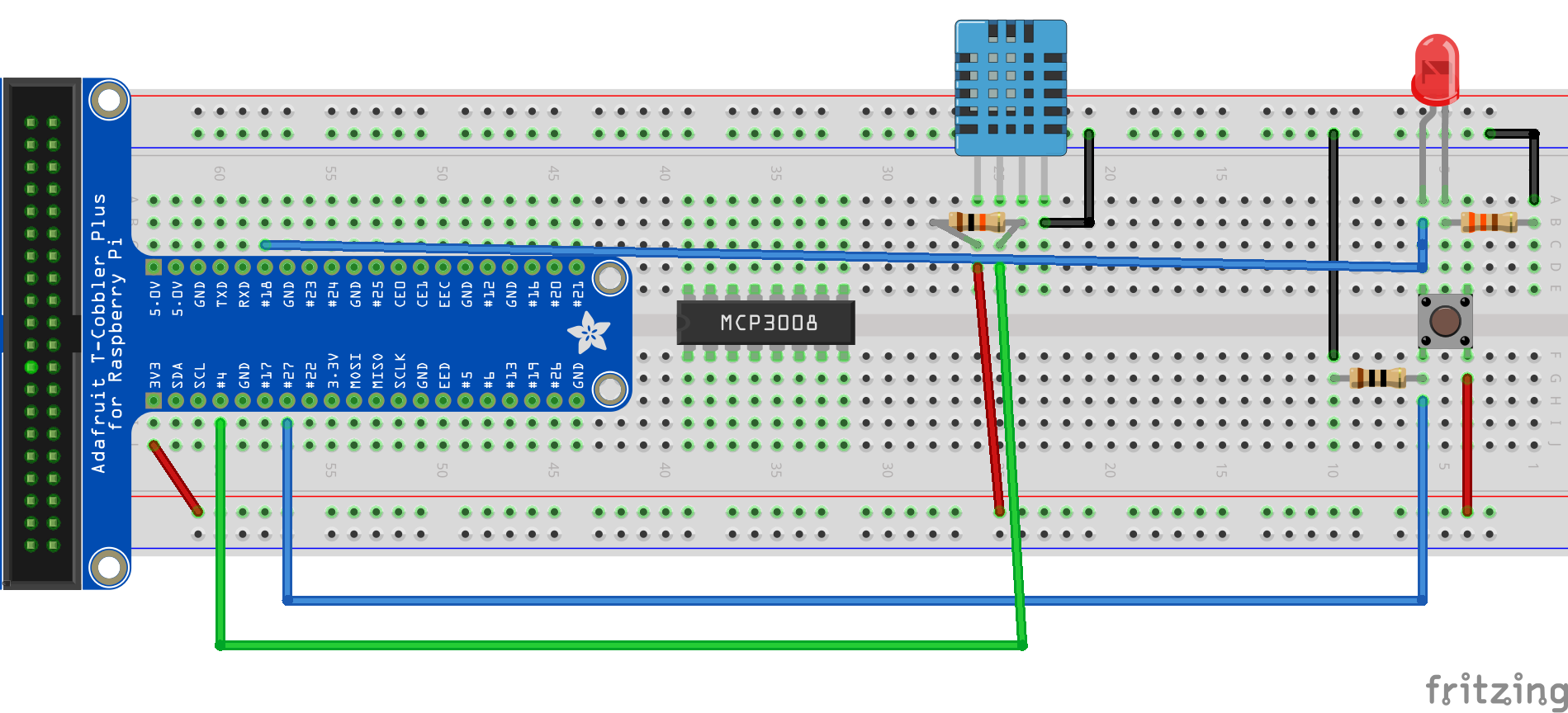


Figure : Fritzing Diagram - master bed room

## 6.1 Connect the components

|  |  |
| --- | --- |
| No. | Task |
| Connect LED and Resistor | |
| a) | Plug in the Red LED into the breadboard as shown. Place the longer end of the LED nearer to the T-Cobbler Kit. |
| b) | Plug in the 330 Ω resistor into the breadboard as shown. One end of the resistor should be in the same column as the short leg of the LED. |
| c) | Using a black cable, connect it one end to GND and another end to top positive (+) power rail |
| d) | **LED**  (i) Using a **blue** cable, connect the long end LED to **BCM18** |
| e) | **Resistor (Orange-Orange-Brown)**  (i) Using a **black** cable, connect the free end of the 330 Ω resistor to **Top positive(+) power rail** |
| Connect DHT11 Sensor | |
| a) | Place the DHT11 Sensor on the breadboard as shown in figure 1. |
| b) | Plug in the 10 Ω resistor into the breadboard as shown in figure 1. |
| c) | Using a red cable, connect it one end to 3.3V and another end to bottom positive (+) power rail |
| d) | **DHT11 Pin 1**  (i) Using a red cable, connect it to **bottom positive (+) power rail** |
| e) | **DHT11 Pin 2**  (i) Using a green cable, connect it to **BCM4** |
| g) | **DHT11 Pin 4**  (i) Using a black cable, connect it to **Top positive (+) power rail** |
|  |  |

# Section 7 – Install Mosquitto

|  |  |
| --- | --- |
| No. | Task |
| a) | Install the Mosquitto broker and clients on your Raspberry Pi with this command  sudo apt‐get install mosquitto mosquitto‐clients |

# Section 8 – AWS IoT Console

|  |  |
| --- | --- |
| No. | Task |
| Sign in to AWS IoT Console | |
| a) | **If you do need have an AWS account, please create one before continuing the following steps.**  Sign in with your AWS console at <https://aws.amazon.com> |
| b) | In the AWS dashboard, type “AWS IoT” to access the AWS IoT service. |
| Create and Register your “Thing” | |
| a) | At the left corner. Click on Registry Things. |
| b) | After clicking the Registry things it will show the follow screen:    Click on the “Register a thing” |
| c) | Type the following into the text box:    Click on the “create thing” |
| d) | At the left corner, Click Security.  After click on the Security. Below screen will show.    Click on the “Create Certificate”  Download all the four file shown below:   * + 1. A certificate for this thing     2. A public Key     3. A private Key     4. A root CA for AWS IoT     After download all the four file. Click “Activate”.  After clicking on “Activate”. Click “Attach a policy” |
| e) | After clicked on “Attach a policy”. Below screen will be shown.  Click on the “Create a new policy”    Type the following into the respective text box:   |  |  | | --- | --- | | Name |  | | Action | Iot:\* | | Resource ARN | \* | | Allow | Checked |     After Creating the policy. Click on back |
| Attach Policy and Thing | |
| a) | In the left corner, Click on Security Certificates. |
| b) | Select the latest Certificates and Click on Action.  Click on “Attach policy” Attach the policy name: And Click on “Attach”  Click on “Attach thing”. Attach the thing name: And Click on “ Attach” |

# Section 9 – Set up AWS DynamoDB

In this section, you will learn how to set up AWS DynamoDB. AWS DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability.

|  |  |
| --- | --- |
| **Search for AWS DynamoDB** | |
| a) | Under the **Service** search for DynamoDB |
| b) | Choose **Create Table** |
| c) | Fill in the fields to match the follwing and create the table  Table name: babyroom  Primary key: babyid  Use default setting |
| d) | After table is being successfully created, you should be able to see the following. Preform the same steps and create table for living room, master room and LED. |
| e) | Overview of the tables:  BabyRoom used to stored data for baby room  LivingRoom used to stored data for living room  MasterRoom used to stored data for master room  LED used to store the status of the LED in the different rooms |

# Section 10 – Set up AWS S3

In this section, you will learn how to set up AWS S3. AWS S3 is a storage for the Internet. It is designed to make web-scale computing easier for developers. Amazon S3 has a simple web services interface that you can use to store and retrieve any amount of data, at any time, at anywhere on the web. It gives any develop access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites.

|  |  |
| --- | --- |
| **Set up for AWS S3** | |
| a) | Under the **Service** search for S3 |
| b) | Select Create Bucket |
| c) | Name: babyroom  Region: US East(Ohio)    Region is based on your account, you can check your region by check the website link |
| d) | Skip this page, just Next |
| e) | Left the setting as default except change the manage public permission to Grant public read access to this bucket. This would allow others to have read access to the bucket. |
| **f)** | Create bucket |
| g) | Overview of the buckets:  Babyroom used to stored images for baby table  Iotlivingroom used to stored images for living room table |

# Section 11 – Create Rule

|  |  |
| --- | --- |
| The AWS IoT rules engine listens for incoming MQTT messages that match a rule. When a matching message is received, the rule takes some action with the data in the MQTT message. | |
| a) | In the AWS IoT console, in the left navigation pane, choose Rule, then ”create a rule” |
| b) | On the Create a rule page, in the Name field, type a name for your rule. In the Description field, type a description for the rule. |
| c) | Scroll down to Message source. Choose the latest version from the Using SQL version dropdown list. In the Attribute field, type \*. This specifies that you want to send the entire MQTT message that triggered the rule. |
| d) | The rules engine uses the topic filter to determine which rules to trigger when an MQTT message is received.  In the Topic filter field, type  “living/sensors”  Leave the condition blank |
| e) | In the Set one or more actions, choose Add action. |
| f) | On the Select an action page, select Split message into multiple columns of a database table |
| g) | On the Table Name select LivingRoom and the IAM role name, choose create a new role. Name the new role as iotca2 and click Create a new role.  After that, choose Add action |
| h) | Last step, choose Create rule and you are done. |
| i) | You will also need to create SNS push notification.  On the Create a rule page, in the Name field, type a name for your rule. In the Description field, type a description for the rule. |
| j) | Same steps as above expect change the Topic filter to baby/alert |
| K) | Choose Send a message as an SNS push notification |
| l) | Change the setting as shown and create the rule |
| m) | Overview of the rules:  LED for led table  LivingRule for livingroom table  MasterRule for masterroom table  BabyRule for babyroom table  Babyalert for the email notification |

# Section 12 – Create IAM User

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources for your users. Your use IAM to control who can use your AWS resources (authentication) and what resources they can use and in what ways (authorization)

|  |  |
| --- | --- |
| a) | Under the Service search for IAM |
| b) | Choose Add User |
| c) | Under User name input your name  For the Access type choose Programmatic access  And click Next permissions |
| d) | Choose Add user to group and choose Create group |  |
| e) | In the Group name input iotca2  For the policy type choose AmazonDynamaDBFullAccess and AmazonS3FullAccess |  |
| f) | You will be redirect back to the Permission page, click Next to proceed to Review Page |  |
| g) | Click Create User and you should receive the success alert.  You will need to download csv file or copy your Access key and secret access key and save in notepad as you will need to use them later for configuration. |  |

# Section 13 – Install the AWS Python SDK

By now, you have completed connecting your device to AWS Cloud. In the next few sections, you will begin to code your app. However, before you can start coding an IoT app on the AWS Cloud, you will need to install the AWS Python software libraries first.

|  |  |
| --- | --- |
| **Install AWS IoT SDK** | |
| a) | Install the AWS Python library with this command  sudo pip install AWSIoTPythonSDK  Install the boto3  sudo pip install boto3 |

# Section 14 - Python Code – Living Room

In this section, you will be creating the python scripts for Living room. You will need to include the 4 files that you had previously downloaded while configuring your “thing”. The MFRC522.py and MFRC522.pyc are the library to enable the LCD screen.

This is the overview of the files that you will need for your Living Room. The images folder is the place where you will be storing your images, where the piCam act as CCTV



## 14.1 buttonAWS.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Create a python script buttonAWS.py with the code below.  Sudo nano ~labs/livingroom/buttonAWS.py |
| b) | This is code you need for the buttonAWS.py file  import sys  import MySQLdb  from gpiozero import LED, Button, MotionSensor, MCP3008  from signal import pause  from threading import Thread  import Adafruit\_DHT  from time import sleep, time, gmtime, strftime  from rpi\_lcd import LCD  import RPi.GPIO as GPIO  import MFRC522  import signal  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  import json  import uuid  import boto3  from boto3.dynamodb.conditions import Key, Attr  from botocore.exceptions import ClientError  # Custom MQTT message callback  def customCallback(client, userdata, message):  print("Received a new message: ")  print(message.payload)  print("from topic: ")  print(message.topic)  print("--------------\n\n")  host = ""  rootCAPath = "rootca.pem"  certificatePath = "certificate.pem.crt"  privateKeyPath = "private.pem.key"  my\_rpi = AWSIoTMQTTClient("basicPubSub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  # Connect and subscribe to AWS IoT  my\_rpi.connect()  my\_rpi.subscribe("sensors/led", 1, customCallback)  sleep(2)  button = Button(27, pull\_up=False)  led = LED(18)  led2 = LED(21)  led3 = LED(20)  def ledON():  led.on()  led2.on()  led3.on()  send\_msg = '{"LEDID": "living","currstat":"ON"}'  send\_msg1 = '{"LEDID": "living1","currstat":"ON"}'  send\_msg2 = '{"LEDID": "living2","currstat":"ON"}'  send\_msg3 = '{"LEDID": "living3","currstat":"ON"}'  my\_rpi.publish("sensors/led",send\_msg,1)  my\_rpi.publish("sensors/led",send\_msg1,1)  my\_rpi.publish("sensors/led",send\_msg2,1)  my\_rpi.publish("sensors/led",send\_msg3,1)    def ledOFF():  led.off()  led2.off()  led3.off()  send\_msg = '{"LEDID": "living","currstat":"OFF"}'  send\_msg1 = '{"LEDID": "living1","currstat":"OFF"}'  send\_msg2 = '{"LEDID": "living2","currstat":"OFF"}'  send\_msg3 = '{"LEDID": "living3","currstat":"OFF"}'  my\_rpi.publish("sensors/led",send\_msg,1)  my\_rpi.publish("sensors/led",send\_msg1,1)  my\_rpi.publish("sensors/led",send\_msg2,1)  my\_rpi.publish("sensors/led",send\_msg3,1)    def detectOFF():  led.off()  led2.off()  led3.off()  def detectON():  led.on()  led2.on()  led3.on()  dynamodb = boto3.resource("dynamodb",aws\_access\_key\_id=#yourKEY,  aws\_secret\_access\_key=#, region\_name='us-east-2')  table = dynamodb.Table('LED')  LEDID = "living"  LEDID1 = "living1"  LEDID2 = "living2"  LEDID3 = "living3"  # Publish to the same topic in a loop forever  loopCount = 0  while True:      dbuuid = str(uuid.uuid1())      DATE = strftime("%Y-%m-%d",gmtime())      TIME = strftime("%H:%M:%S",gmtime())      status= led.is\_lit      if status == True:     button.when\_pressed = ledOFF        if status == False:     button.when\_pressed = ledON        try:       response1 = table.get\_item(       Key={       'LEDID': LEDID1         }       )      except ClientError as e:       print(e.response1['Error']['Message'])      else:       living1 = str(response1['Item']['currstat'])       print("GetItem succeeded:")          if living1 == "OFF":         led.off()    else:          led.on()  # ----------------------------------------------------- #      try:       response2 = table.get\_item(       Key={       'LEDID': LEDID2         }       )      except ClientError as e:       print(e.response2['Error']['Message'])      else:       living2 = str(response2['Item']['currstat'])       print("GetItem succeeded:")      if living2 == "OFF":         led2.off()    else:          led2.on()    # ----------------------------------------------------- #      try:       response3 = table.get\_item(       Key={       'LEDID': LEDID3         }       )      except ClientError as e:       print(e.response3['Error']['Message'])      else:       living3 = str(response3['Item']['currstat'])       print("GetItem succeeded:")      if living3 == "OFF":         led3.off()    else:          led3.on()    \*\*\*aws\_access\_key\_id & aws\_secret\_access\_key is the key downloaded from IAM user |
| c) | Run the program in the Terminal  Sudo python buttonAWS.py |
| d) | If everything runs correctly, you should see similar output as shown. The output means that the button is inside while loop, which shows that the button is being successfully triggered. |
| e) | You must ensure that the 4 files downloaded from ”thing” is in the same level as the python script in order for the code to run. |

## 14.2 lcd.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Create a python script lcd.py with the code below.  Sudo nano ~labs/livingroom/lcd.py |
| b) | This is the code for **lcd.py**  # Import SDK packages  import sys  import Adafruit\_DHT  from time import sleep, time, gmtime, strftime  from rpi\_lcd import LCD  import RPi.GPIO as GPIO  import MFRC522  import signal  from signal import pause  import picamera  from gpiozero import LED, Button, MotionSensor, MCP3008  from threading import Thread  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  import json  import uuid  loopCount = 0  pin = 4  lcd = LCD()  prev\_uid = None  continue\_reading = True  def end\_read(signal,frame):     global continue\_reading     print ("Ctrl+C captured, ending read.")     continue\_reading = False     GPIO.cleanup()  signal.signal(signal.SIGINT, end\_read)  mfrc522 = MFRC522.MFRC522()  print "Welcome to the MFRC522 data read example"  print "Press Ctrl-C to stop."  lcd.text('Welcome to ', 1)  lcd.text('Smart Home', 2)  OwnerOne = [136, 4, 100, 38, 206]  OwnerTwo = [136, 4, 133, 233, 224]  Roomstatus = 0  entry = []  def checkAuth(uid):  global Roomstatus  if uid == OwnerOne:  owner = "xuchao"  if "xuchao" in entry:  entry.remove(owner)  Roomstatus -= 1  lcd.text('Good bye', 1)  lcd.text('Take care!', 2)  else:  entry.append(owner)  Roomstatus += 1  lcd.text('Welcome home', 1)  lcd.text('Xu Chao', 2)  elif uid == OwnerTwo:  owner2 = "kaiyuan"  if "kaiyuan" in entry:  entry.remove(owner2)  Roomstatus -= 1  lcd.text('Good bye', 1)  lcd.text('Take care!', 2)  else:  entry.append(owner2)  Roomstatus += 1  lcd.text('Welcome home', 1)  lcd.text('Kai Yuan', 2)  print(Roomstatus)  print("owner inside smart home")  print(entry)  while continue\_reading:  (status, TagType) = mfrc522.MFRC522\_Request(mfrc522.PICC\_REQIDL)  if status == mfrc522.MI\_OK:  (status, uid) = mfrc522.MFRC522\_Anticoll()  if uid != prev\_uid:    if uid == OwnerOne or uid ==OwnerTwo:    checkAuth(uid)  sleep(2)  lcd.text('Welcome to ', 1)  lcd.text('Smart Home', 2) |
| c) | Run the program in the Terminal  Sudo python lcd.py |
| d) | If everything runs correctly, you should see similar output as shown. It will generate new record when you tap your UUID card on the reader. |

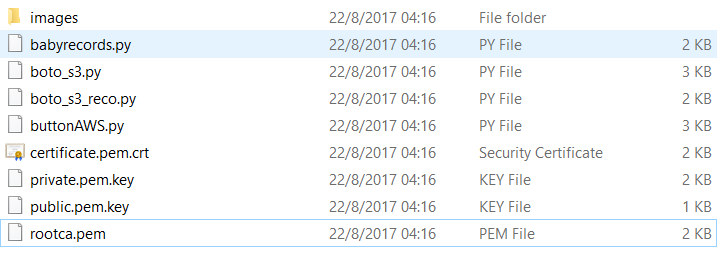
## 14.3 records.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Create a python script records.py with the code below.  Sudo nano ~labs/livingroom/records.py |
| b) | This is the code for **records.py**  # Import SDK packages  import sys  import Adafruit\_DHT  from time import sleep, time, gmtime, strftime  from rpi\_lcd import LCD  import RPi.GPIO as GPIO  import MFRC522  import signal  from signal import pause  from picamera import PiCamera  from gpiozero import LED, Button, MotionSensor, MCP3008  from threading import Thread  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  import json  import uuid  import boto3  import botocore  from datetime import datetime,date  # Custom MQTT message callback  def customCallback(client, userdata, message):  print("Received a new message: ")  print(message.payload)  print("from topic: ")  print(message.topic)  print("--------------\n\n")  host = ""  rootCAPath = "rootca.pem"  certificatePath = "certificate.pem.crt"  privateKeyPath = "private.pem.key"  my\_rpi = AWSIoTMQTTClient("basicPubSub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  my\_rpi.connect()  my\_rpi.subscribe("living/sensors", 1, customCallback)  pin = 4  bucket\_name = 'iotlivingroom' # replace with your own unique bucket name  location={'LocationConstraint':'us-east-2'}  pir = MotionSensor(26, sample\_rate=5,queue\_len=1)  def takePhoto(file\_path,file\_name):  with PiCamera() as camera:  full\_path = file\_path+'/'+file\_name  camera.capture(full\_path)  def uploadToS3(file\_path,file\_name,bucket\_name,location):  # Create an S3 resource manually  s3 = boto3.resource('s3',  aws\_access\_key\_id=# Your Key,  aws\_secret\_access\_key=#YourKey)  exists=True  try:  s3.meta.client.head\_bucket(Bucket=bucket\_name)  except botocore.exceptions.ClientError as e:  error\_code = int(e.response['Error']['Code'])  if error\_code == 404:  exists = False  if exists == False:  s3.create\_bucket(Bucket=bucket\_name,CreateBucketConfiguration=location)  # Upload a new file  full\_path=file\_path+"/"+file\_name  s3.Object(bucket\_name, file\_name).put(Body=open(full\_path,'rb'))    object\_acl = s3.ObjectAcl('iotlivingroom',file\_name)  response=object\_acl.put(ACL='public-read')  print("File uploaded")  #s3.upload\_file(full\_path, bucket\_name, file\_name)  while True:      timestring = strftime("%Y-%m-%dT%H:%M:%S",gmtime())   file\_path = '/home/pi/labs/livingroom/images'   file\_name = 'photo\_'+timestring+'.jpg'  takePhoto(file\_path,file\_name)    uploadToS3(file\_path,file\_name,bucket\_name,location)      humidity, temperature = Adafruit\_DHT.read\_retry(11, pin)      dbuuid = uuid.uuid1()    DATE = strftime("%Y-%m-%d",gmtime())    TIME = strftime("%H:%M:%S",gmtime())    TEMP = "{:.1f}".format(temperature)    HUMI = "{:.1f}".format(humidity)      send\_msg = '{"livingID": "' + str(dbuuid) + '","Date": "' + DATE + '","Time": "' + TIME + '","Temperature": "' + TEMP + '","Humidity": "' + HUMI + '"}'    my\_rpi.publish("living/sensors",send\_msg,1)    PiCamera().close()    sleep(5) |
| c) | Run the program in the Terminal  Sudo python records.py |
| d) | If everything runs correctly, you should see similar output as shown. Let it run for a few rounds before pressing Ctrl-C to terminate the program.  The output “File uploaded” means the image files has being successfully uploaded to the AWS S3 and the message has being uploaded to the DynamoDB |

# Section 15 - Python Code – Baby Room

In this section, you will be creating the python scripts for Baby room. You will need to include the 4 files that you had previously downloaded while configuring your “thing”.

This is the overview of the files that you will need for your Baby Room. The images folder is the place to store images, where the images is being captured when there is motion.



## 15.1 babyrecords.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Create a python script babyrecords.py with the code below.  Sudo nano ~labs/babyroom/babyrecords.py |
| b) | This is code you need for the babyrecords.py file  from gpiozero import MCP3008, MotionSensor, LED, Button  from time import sleep, time, gmtime, strftime  import sys  import Adafruit\_DHT  import RPi.GPIO as GPIO  from signal import pause  import picamera  from threading import Thread  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  import json  import uuid  # Custom MQTT message callback  def customCallback(client, userdata, message):  print("Received a new message: ")  print(message.payload)  print("from topic: ")  print(message.topic)  print("--------------\n\n")  host = "  rootCAPath = "rootca.pem"  certificatePath = "certificate.pem.crt"  privateKeyPath = "private.pem.key"  my\_rpi = AWSIoTMQTTClient("basicPubSub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  my\_rpi.connect()  my\_rpi.subscribe("baby/sensors", 1, customCallback)  mcp3008 = MCP3008(channel=0) # light  pir = MotionSensor(26, sample\_rate=5,queue\_len=1) # motion  pin = 4 #tem & humidity  humidity, temperature = Adafruit\_DHT.read\_retry(11,pin)  update = True  while update:  print("MCP3008 raw value={}".format(mcp3008.value))  sensor\_value = (1024/mcp3008.value)%1024  sensor\_value = str(round(sensor\_value))  print(sensor\_value)  dbuuid = uuid.uuid1()  DATE = strftime("%Y-%m-%d",gmtime())  TIME = strftime("%H:%M:%S",gmtime())  TEMP = "{:.1f}".format(temperature)  HUMI = "{:.1f}".format(humidity)  send\_msg = '{"babyID": "' + str(dbuuid) + '","Date": "' + DATE + '","Time": "' + TIME + '","Temperature": "' + TEMP + '","Humidity": "' + HUMI + '","Light":"' + sensor\_value + '"}'  my\_rpi.publish("baby/sensors",send\_msg,1)  sleep(5) |
| c) | Run the program in the Terminal  Sudo python babyrecords.py |
| d) | If everything runs correctly, you should see similar output as shown. These values will be uploaded to the DynamoDB. |
| e) | You must ensure that the 4 files downloaded from ”thing” is in the same level as the python script in order for the code to run. |

## 15.2 buttonAWS.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Create a python script buttonAWS.py with the code below.  Sudo nano ~labs/babyroom/buttonAWS.py |
| b) | This is the code for **buttonAWS.py**  import sys  import MySQLdb  from gpiozero import LED, Button, MotionSensor, MCP3008  from signal import pause  from threading import Thread  import Adafruit\_DHT  from time import sleep, time, gmtime, strftime  from rpi\_lcd import LCD  import RPi.GPIO as GPIO  import signal  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  import json  import uuid  import boto3  from boto3.dynamodb.conditions import Key, Attr  from botocore.exceptions import ClientError  # Custom MQTT message callback  def customCallback(client, userdata, message):  print("Received a new message: ")  print(message.payload)  print("from topic: ")  print(message.topic)  print("--------------\n\n")  host = ""  rootCAPath = "rootca.pem"  certificatePath = "certificate.pem.crt"  privateKeyPath = "private.pem.key"  my\_rpi = AWSIoTMQTTClient("basicPubSub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  # Connect and subscribe to AWS IoT  my\_rpi.connect()  my\_rpi.subscribe("sensors/led", 1, customCallback)  sleep(2)  button = Button(27, pull\_up=False)  led = LED(18)  def ledON():  led.on()  send\_msg = '{"LEDID": "baby","currstat":"ON"}'    my\_rpi.publish("sensors/led",send\_msg,1)      def ledOFF():  led.off()  send\_msg = '{"LEDID": "baby","currstat":"OFF"}'  my\_rpi.publish("sensors/led",send\_msg,1)    def detectOFF():  led.off()  def detectON():  led.on()  dynamodb = boto3.resource("dynamodb",aws\_access\_key\_id=YourKEY,  aws\_secret\_access\_key=YourKey, region\_name='us-east-2')  table = dynamodb.Table('LED')  LEDID = "baby"    # Publish to the same topic in a loop forever  loopCount = 0  while True:      dbuuid = str(uuid.uuid1())      DATE = strftime("%Y-%m-%d",gmtime())      TIME = strftime("%H:%M:%S",gmtime())      status= led.is\_lit      if status == True:     button.when\_pressed = ledOFF        if status == False:     button.when\_pressed = ledON        try:       response = table.get\_item(       Key={       'LEDID': LEDID         }       )      except ClientError as e:       print(e.response['Error']['Message'])      else:       living = str(response['Item']['currstat'])       print("GetItem succeeded:")        if living == "OFF":         detectOFF()    else:          detectON()  \*\*\*aws\_access\_key\_id & aws\_secret\_access\_key is the key downloaded from IAM user |
| c) | Run the program in the Terminal  Sudo python buttonAWS.py |
| d) | If everything runs correctly, you should see similar output as shown. The output means that the button is inside while loop, which shows that the button is being successfully triggered. |

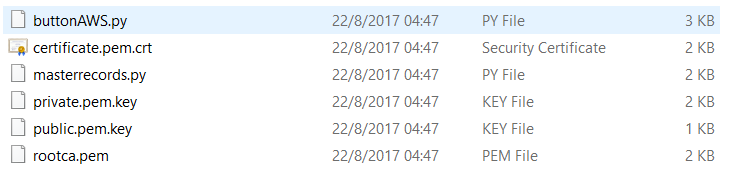
## 15.3 boto\_s3.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Create a python script boto\_s3.py with the code below.  Sudo nano ~labs/babyroom/boto\_s3.py |
| b) | This is the code for **boto\_s3.py**  import boto3  import botocore  from picamera import PiCamera  from time import sleep,time  import time  from gpiozero import MotionSensor  from datetime import datetime, date  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  def customCallback(client, userdata, message):  print("Received a new message: ")  print(message.payload)  print("from topic: ")  print(message.topic)  print("--------------\n\n")  host = "#"  rootCAPath = "rootca.pem"  certificatePath = "certificate.pem.crt"  privateKeyPath = "private.pem.key"  my\_rpi = AWSIoTMQTTClient("basicPubSub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  my\_rpi.connect()  my\_rpi.subscribe("baby/alert", 1, customCallback)  bucket\_name = 'babyroom' # replace with your own unique bucket name  location={'LocationConstraint':'us-east-2'}  pir = MotionSensor(26, sample\_rate=5,queue\_len=1)  def takePhoto(file\_path,file\_name):  with PiCamera() as camera:  full\_path = file\_path+'/'+file\_name  camera.capture(full\_path)  def uploadToS3(file\_path,file\_name,bucket\_name,location):  # Create an S3 resource manually  s3 = boto3.resource('s3',  aws\_access\_key\_id=#yourkey,  aws\_secret\_access\_key=#yourkey)  exists=True  try:  s3.meta.client.head\_bucket(Bucket=bucket\_name)  except botocore.exceptions.ClientError as e:  error\_code = int(e.response['Error']['Code'])  if error\_code == 404:  exists = False  if exists == False:  s3.create\_bucket(Bucket=bucket\_name,CreateBucketConfiguration=location)  # Upload a new file  full\_path=file\_path+"/"+file\_name  s3.Object(bucket\_name, file\_name).put(Body=open(full\_path,'rb'))  object\_acl = s3.ObjectAcl('babyroom',file\_name)  response=object\_acl.put(ACL='public-read')  print("File uploaded")  #s3.upload\_file(full\_path, bucket\_name, file\_name)  while True:  timestring = time.strftime("%Y-%m-%dT%H:%M:%S",time.gmtime())  file\_path = '/home/pi/labs/babyroom/images'  file\_name = 'photo\_'+timestring+'.jpg'  if pir.wait\_for\_motion():  print("Motion detected... take photo")  takePhoto(file\_path,file\_name)  uploadToS3(file\_path,file\_name,bucket\_name,location)  send\_msg = 'A heavy motion detected in baby room'  my\_rpi.publish("baby/alert",send\_msg,1)  sleep(3)  else:  print("Room is quiet now...Baby sleeping")  sleep(5) |
| c) | Run the program in the Terminal  Sudo python boto\_s3.py |
| d) | If everything runs correctly, you should see similar output as shown. Once motion is being detected, it will take a photo and upload to S3 and sent an email to notify the user of the motion. |

# Section 16 - Python Code – Master Room

In this section, you will be creating the python scripts for Master room. You will need to include the 4 files that you had previously downloaded while configuring your “thing”.

This is the overview of the files that you will need for your Master Room.



## 16.1 masterrecords.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Create a python script masterrecords.py with the code below.  Sudo nano ~labs/masterroom/masterrecords.py |
| b) | This is code you need for the masterrecords.py file  from gpiozero import MCP3008, MotionSensor, LED, Button  from time import sleep, time, gmtime, strftime  import sys  import Adafruit\_DHT  import RPi.GPIO as GPIO  from signal import pause  from threading import Thread  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  import json  import uuid  # Custom MQTT message callback  def customCallback(client, userdata, message):  print("Received a new message: ")  print(message.payload)  print("from topic: ")  print(message.topic)  print("--------------\n\n")  host = "a "  rootCAPath = "rootca.pem"  certificatePath = "certificate.pem.crt"  privateKeyPath = "private.pem.key"  my\_rpi = AWSIoTMQTTClient("basicPubSub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  my\_rpi.connect()  my\_rpi.subscribe("master/sensors", 1, customCallback)  sleep(2)  pin = 4 #tem & humidity  update = True  while update:  dbuuid = uuid.uuid1()  DATE = strftime("%Y-%m-%d",gmtime())  TIME = strftime("%H:%M:%S",gmtime())  humidity, temperature = Adafruit\_DHT.read\_retry(11,pin)  TEMP = "{:.1f}".format(temperature)  HUMI = "{:.1f}".format(humidity)  send\_msg = '{"masterID": "' + str(dbuuid) + '","Date": "' + DATE + '","Time": "' + TIME + '","Temperature": "' + TEMP + '","Humidity": "' + HUMI + '"}'  my\_rpi.publish("master/sensors",send\_msg,1)  sleep(5) |
| c) | Run the program in the Terminal  Sudo python masterrecords.py |
| d) | If everything runs correctly, you should see similar output as shown. These values will be uploaded to the DynamoDB. |
| e) | You must ensure that the 4 files downloaded from ”thing” is in the same level as the python script in order for the code to run. |

## 16.2 buttonAWS.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Create a python script buttonAWS.py with the code below.  Sudo nano ~labs/masterroom/buttonAWS.py |
| b) | This is the code for **buttonAWS.py**  import sys  from gpiozero import LED, Button, MotionSensor, MCP3008  from signal import pause  from threading import Thread  import Adafruit\_DHT  from time import sleep, time, gmtime, strftime  from rpi\_lcd import LCD  import RPi.GPIO as GPIO  import signal  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  import json  import uuid  import boto3  from boto3.dynamodb.conditions import Key, Attr  from botocore.exceptions import ClientError  # Custom MQTT message callback  def customCallback(client, userdata, message):  print("Received a new message: ")  print(message.payload)  print("from topic: ")  print(message.topic)  print("--------------\n\n")  host = ""  rootCAPath = "rootca.pem"  certificatePath = "certificate.pem.crt"  privateKeyPath = "private.pem.key"  my\_rpi = AWSIoTMQTTClient("basicPubSub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  # Connect and subscribe to AWS IoT  my\_rpi.connect()  my\_rpi.subscribe("sensors/led", 1, customCallback)  sleep(2)  button = Button(27, pull\_up=False)  led = LED(18)  def ledON():  led.on()    send\_msg = '{"LEDID": "master","currstat":"ON"}'    my\_rpi.publish("sensors/led",send\_msg,1)      def ledOFF():  led.off()  send\_msg = '{"LEDID": "master","currstat":"OFF"}'    my\_rpi.publish("sensors/led",send\_msg,1)      def detectOFF():  led.off()  def detectON():  led.on()  dynamodb = boto3.resource("dynamodb",aws\_access\_key\_id=#yourkey,  aws\_secret\_access\_key=#yourkey, region\_name='us-east-2')  table = dynamodb.Table('LED')  LEDID = "master"    # Publish to the same topic in a loop forever  loopCount = 0  while True:      dbuuid = str(uuid.uuid1())      DATE = strftime("%Y-%m-%d",gmtime())      TIME = strftime("%H:%M:%S",gmtime())      status= led.is\_lit      if status == True:     button.when\_pressed = ledOFF        if status == False:     button.when\_pressed = ledON        try:       response = table.get\_item(       Key={       'LEDID': LEDID         }       )      except ClientError as e:       print(e.response['Error']['Message'])      else:       living = str(response['Item']['currstat'])       print("GetItem succeeded:")        if living == "OFF":         detectOFF()    else:          detectON()  \*\*\*aws\_access\_key\_id & aws\_secret\_access\_key is the key downloaded from IAM user |
| c) | Run the program in the Terminal  Sudo python buttonAWS.py |
| d) | If everything runs correctly, you should see similar output as shown. The output means that the button is inside while loop, which shows that the button is being successfully triggered. |

# Section 17 – PHP configuration

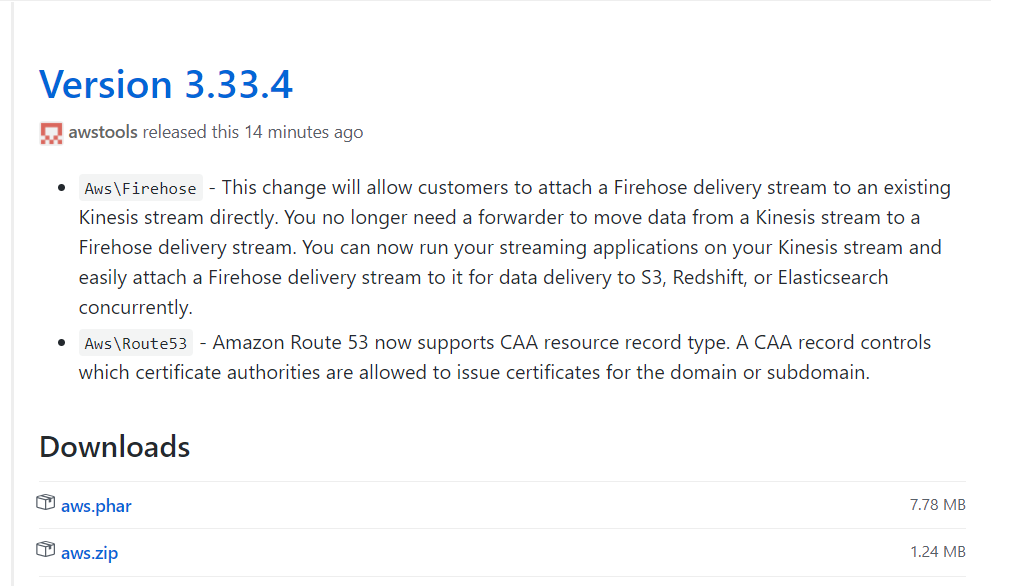
In this section, you will set up necessary files and configuration for use the AWS Cloud features on PHP.

XAMPP Apache server is recommended for your PHP project. Link to download XAMPP: <https://www.apachefriends.org/index.html>

Alternatively, you can server the PHP on raspberry pi. However, it may not be very powerful as the pi might take more time to load the web app.

Firstly, download the newest version AWS SDK for PHP in this link:

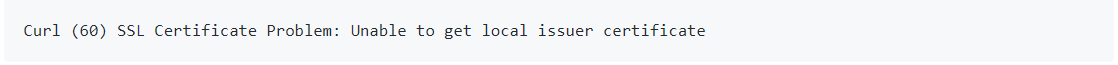
<https://github.com/aws/aws-sdk-php/releases>



Extract the aws.zip file and place the aws folder in the same directory of your PHP project.

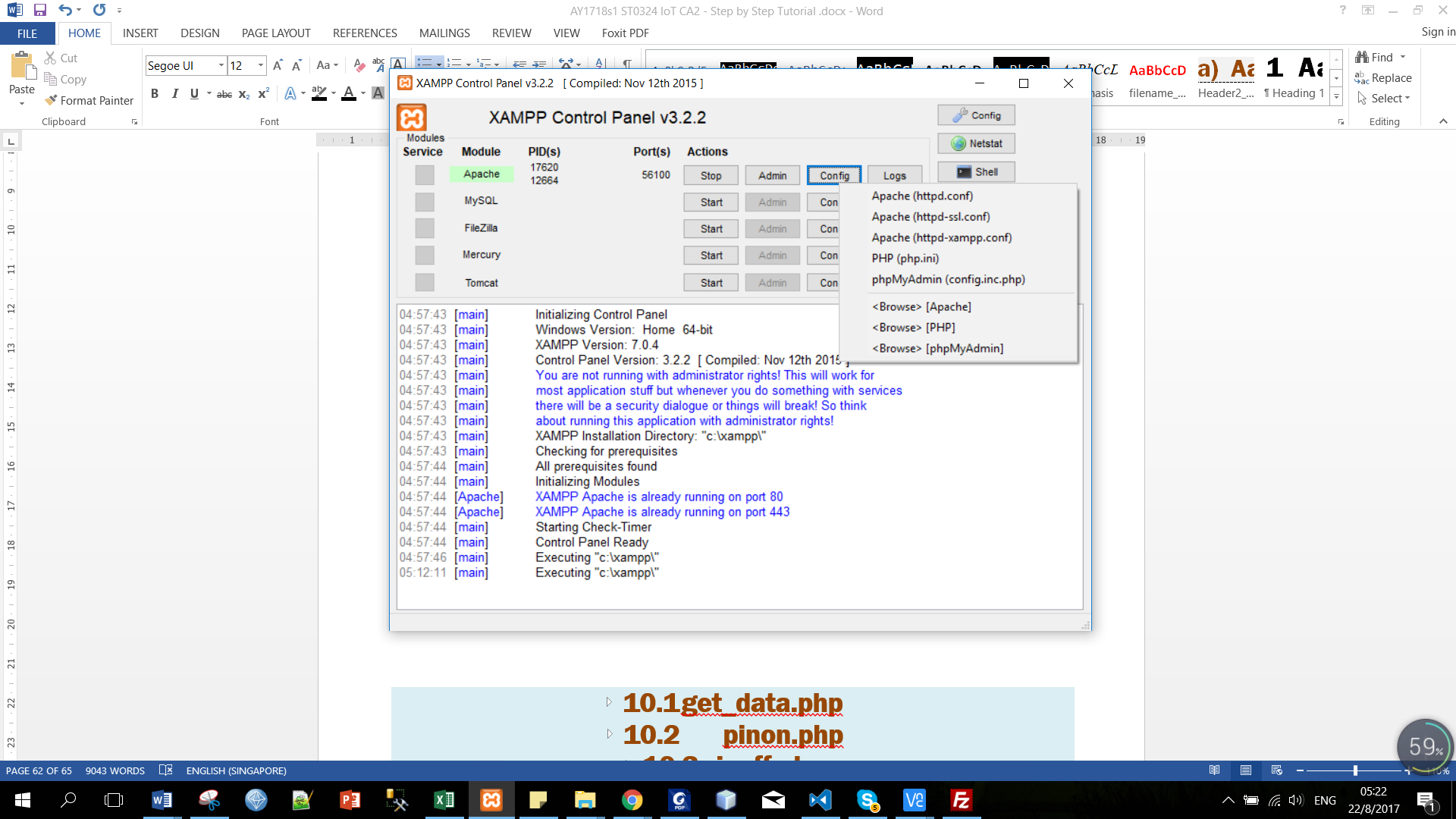
Setting up cURL SSL/TLS Certificate Authority Certificates for PHP

If your system is not correctly set up with SSL/TLS Certificate Authority (CA) certificates, you might get the following error when trying to build connection with cloud services:



Thus, there might be a need to download a configuration file to enable to service. Download the cacert.pem file in this link: <https://github.com/bolt/docs/blob/3.3/docs/howto/curl-ca-certificates.md>

* Place the file under the PHP folder e.g. *C:\php\extras\ssl\cacert.pem*
* Edit the php.ini file and change the curl.cainfo parameter, e.g. curl.cainfo = "C:\php\extras\ssl\cacert.pem". If you are using XAMPP server, you can access the file from the panel.



Below will show configuration PHP files for establishing and retrieve data from AWS DynamoDB. For other PHP files please refer to the web folder.

## config.php

|  |  |
| --- | --- |
| No. | Task |
| a) | The following code is setting up the configuration of the AWS DynamoDB connection with PHP. |
| b) | This is the code for **config.php**  <?php  require 'aws/aws-autoloader.php';  date\_default\_timezone\_set('UTC');  // use class  use Aws\DynamoDb\Exception\DynamoDbException;  // create new connection  $sdk = new Aws\Sdk([  'version' => 'latest',  'region' => 'us-east-2',  'credentials' => [  'key' =>#,  'secret' =>#,  ],  ]);  //function for scan all rows in the assigned table  function scanAllData($dynamodb,$table){  $result = $dynamodb->scan(array(  'TableName' => $table,    'Select' => 'ALL\_ATTRIBUTES'  )    );  return $result["Items"];  }  ?> |

## alldata.php

|  |  |
| --- | --- |
| No. | Task |
| a) | The following code is basic set up to get all values from three different tables we created in the AWS DynamoDB. The result will produce 3 asccociative array in time order. |
| b) | This is the code for **alldata.php**  <?php  require "config.php";  $dynamodb = $sdk->createDynamoDb();  $getobj = scanAllData($dynamodb, 'LivingRoom');  $getobj2 = scanAllData($dynamodb, 'MasterRoom');  $getobj3 = scanAllData($dynamodb, 'BabyRoom');  foreach ($getobj as $key => $row) {  $volume[$key] = $row['Time']['S'];    }  array\_multisort($volume, SORT\_ASC, $getobj);  foreach ($getobj2 as $key => $row) {  $volume2[$key] = $row['Time']['S'];    }  array\_multisort($volume2, SORT\_ASC, $getobj2);  foreach ($getobj3 as $key => $row) {  $volume3[$key] = $row['Time']['S'];    }  array\_multisort($volume3, SORT\_ASC, $getobj3);  ?> |

## chartdata.php

|  |  |
| --- | --- |
| No. | Task |
| a) | The following code is getting the newest record of each table in DynamoDB and used for real time updating the charts generated in the website. |
| b) | This is the code for **chartdata.php**  <?php  require "config.php";  $name="";  $arr=array();  if($\_GET['id']=='living'){  $name="LivingRoom";  $dynamodb = $sdk->createDynamoDb();  $getobj = scanAllData($dynamodb,$name);  foreach ($getobj as $key => $row) {  $volume[$key] = $row['Time']['S'];    }  array\_multisort($volume, SORT\_ASC, $getobj);  $last =count($getobj)-1;  array\_push($arr, $getobj[$last]['Time']['S'],$getobj[$last]['Humidity']['S'],$getobj[$last]['Temperature']['S']);  }else if($\_GET['id']=='master'){    $name="MasterRoom";  $dynamodb = $sdk->createDynamoDb();  $getobj = scanAllData($dynamodb,$name);  foreach ($getobj as $key => $row) {  $volume[$key] = $row['Time']['S'];    }  array\_multisort($volume, SORT\_ASC, $getobj);  $last =count($getobj)-1;  array\_push($arr, $getobj[$last]['Time']['S'],$getobj[$last]['Humidity']['S'],$getobj[$last]['Temperature']['S']);  }else if($\_GET['id']=='baby'){    $name="BabyRoom";  $dynamodb = $sdk->createDynamoDb();  $getobj = scanAllData($dynamodb,$name);  foreach ($getobj as $key => $row) {  $volume[$key] = $row['Time']['S'];    }  array\_multisort($volume, SORT\_ASC, $getobj);  $last =count($getobj)-1;  array\_push($arr, $getobj[$last]['Time']['S'],$getobj[$last]['Humidity']['S'],$getobj[$last]['Temperature']['S'],$getobj[$last]['Light']['S']);  }  echo json\_encode($arr);  ?> |

## JavaScript Libraries used

|  |  |
| --- | --- |
| No. | Task |
| a) | If you interested on the JavaScript Libraries that has been used, here is the list. |
| b) | * **jQuery library** * **Bootstrap** * **Amcharts** * **Datatables** |

# Section 18 - Test the codes in browser

In this section, you will test out the working application and the expected results.

## 18.1 Run buttonAWS.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Change to masterroom/livingroom/babyroom directory  Cd labs/masterroom |
| b) | Run the buttonAWS.py  Sudo python buttonAWS.py |
| c) | You should see the file starting to run and you should able to control the LED from the application  C:\Users\CJ\Downloads\3b646919-ba58-4585-9c0b-c35d5ece2d88.jpg |

## 18.2 Run records.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Change to masterroom/livingroom/babyroom directory  Cd labs/masterroom |
| b) | Run the masterrecords.py / records.py / babyrecords.py  Sudo python masterrecords.py |
| c) | After the file starts to run and the output appears, the application should reflects the same data. And the records should also reflects on the history page |

## 18.3 Run lcd.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Change to livingroom directory  Cd labs/livingroom |
| b) | If everything goes well, you should see output similar to that below in your terminal and LCD display.    C:\Users\CJ\Downloads\a00ff4ed-6ce2-4a0c-b2f5-64af48f72187.jpg  C:\Users\CJ\Downloads\8d779c9c-2df4-4952-a347-afeb6f3856a9.jpg C:\Users\CJ\Downloads\8c9dd472-5fc4-4e28-9744-a6095f942f58.jpg |
|  |  |

## 18.4 Run boto\_s3.py

|  |  |
| --- | --- |
| No. | Task |
| a) | Change to babyroom directory  Cd labs/babyroom |
| b) | Run the boto\_s3.py  Sudo python boto\_s3.py |
| c) | If everything goes well, you should be able to see the following output in your command prompt, as well as the application picture page and received an notification on your email. |

GitHub Link: https://github.com/mlggg/iotca2

**-- End of CA2 Step-by-step tutorial --**