|  |
| --- |
| Jan Fontanosa, Vyacheslav Perepelytsya, and Maasha Maheson |
| HVAC |
| Internet of Things Capstone Project |

|  |
| --- |
| Humber College  4-23-2018 |

# Declaration of Joint Authorship

We, Jan Fontanosa, Vyacheslav Perepelytsya, and Maasha Maheson, confirm that this work submitted for assessment is our own and is expressed in our own words. Any uses made within it of the works of any other author, in any form (ideas, equations, figures, texts, tables, programs), are properly acknowledged at the point of use. A list of the references used is included.

# Proposal

2018-02-05

***Proposal for the development of HVAC***

Prepared by Jan Fontanosa, Vyacheslav Perepelytsya, and Maasha Maheson  
*Computer Engineering Technology Students*https://github.com/fntj0052/HVAC

**Executive Summary**

As a student in the Computer Engineering Technology program, I will be integrating the knowledge and skills I have learned from our program into this Internet of Things themed capstone project. This proposal requests the approval to build the hardware portion that will connect to a database as well as to a mobile device application. The internet connected hardware will include a custom PCB with the following sensors and actuators: touch sensor, moisture sensor, LCD touchscreen, and sound sensor. The database will store Operational status/condition and maintenance reminder. The mobile device functionality will include the ability to toggle operational mode and set maintenance schedule and will be further detailed in the mobile application proposal. I will be collaborating with the following company/department Humber Greenhouse. In the winter semester I plan to form a group with the following students, who are also building similar hardware this term and working on the mobile application with me Jan Fontanosa, Vyacheslav Perepelytsya, and Maasha Maheson. The hardware will be completed in CENG 317 Hardware Production Techniques independently and the application will be completed in CENG 319 Software Project. These will be integrated together in the subsequent term in CENG 355 Computer Systems Project as a member of a 2 or 3 student group.

**Background**

The problem solved by this project is creating and improving upon the monitoring of Heating, Ventilation and Air Conditioning (HVAC) systems with a user-friendly interface, with the ability to remotely control the system activities using a mobile device, and to fetch stored information on the system's condition from a cloud database.. A bit of background about this topic is HVAC systems are useful in all kinds of building applications: a smarter system can provide significant energy and financial savings while scheduling usage and allowing more granular control for systems used in specific applications (a HVAC system used to monitor an industrial refrigerator room will require different settings in comparison to one used in a residential building). By providing the ability for remote control using an Internet of Things(IoT)-based HVAC system, administrators of the system can ensure that the system is working as intended and can administrate changes to the system in a secure manner..

Existing products on the market include [1]. I have searched for prior art via Humber’s IEEE subscription selecting “My Subscribed Content”[2] and have found and read [3] which provides insight into similar efforts.

In the Computer Engineering Technology program we have learned about the following topics from the respective relevant courses:

* Java Docs from CENG 212 Programming Techniques In Java,
* Construction of circuits from CENG 215 Digital And Interfacing Systems,
* Rapid application development and Gantt charts from CENG 216 Intro to Software Engineering,
* Micro computing from CENG 252 Embedded Systems,
* SQL from CENG 254 Database With Java,
* Web access of databases from CENG 256 Internet Scripting; and,
* Wireless protocols such as 802.11 from TECH152 Telecom Networks.

This knowledge and skill set will enable me to build the subsystems and integrate them together as my capstone project.

**Methodology**

This proposal is assigned in the first week of class and is due at the beginning of class in the second week of the fall semester. My coursework will focus on the first two of the 3 phases of this project:  
 Phase 1 Hardware build.  
 Phase 2 System integration.  
 Phase 3 Demonstration to future employers.

*Phase 1 Hardware build*

The hardware build will be completed in the fall term. It will fit within the CENG Project maximum dimensions of 12 13/16" x 6" x 2 7/8" (32.5cm x 15.25cm x 7.25cm) which represents the space below the tray in the parts kit. The highest AC voltage that will be used is 16Vrms from a wall adaptor from which +/- 15V or as high as 45 VDC can be obtained. Maximum power consumption will be 20 Watts.

*Phase 2 System integration*

The system integration will be completed in the fall term.

*Phase 3 Demonstration to future employers*

This project will showcase the knowledge and skills that I have learned to potential employers.

The brief description below provides rough effort and non-labour estimates respectively for each phase. A Gantt chart will be added by week 3 to provide more project schedule details and a more complete budget will be added by week 4. It is important to start tasks as soon as possible to be able to meet deadlines.

No additional planned purchases

**Concluding remarks**

This proposal presents a plan for providing an IoT solution for for a user-friendly, Internet of Things-based HVAC system. This is an opportunity to integrate the knowledge and skills developed in our program to create a collaborative IoT capstone project demonstrating my ability to learn how to support projects such as the initiative described by [3]. I request approval of this project.

# Abstract

This technical report is about the build progress and other documentation about our Internet of Things (IoT)-based Heating, Ventilation, and Air Conditioning (HVAC) monitoring system. This HVAC monitoring system measures the temperature, moisture, and sound generated from the system comparing them to the user’s specifications. The user can optimise the system in regards to their needs. The system is eco-friendly and optimizes energy and financial savings in a residential, commercial or industrial environment. These smart HVAC monitoring systems increase the efficiency of the working and production environment. This technical report also shows whether the HVAC monitoring system can be a built as a functional project.

Table of Contents

[Declaration of Joint Authorship iii](#_Toc512258461)

[Proposal v](#_Toc512258462)

[Abstract xi](#_Toc512258463)

[List of Figures xv](#_Toc512258464)

[1. Introduction 1](#_Toc512258465)

[2. Project Description 3](#_Toc512258466)

[2.1 Project Requirements Specification 3](#_Toc512258467)

[2.1.1 Purpose 3](#_Toc512258468)

[2.1.2 Document Conventions 3](#_Toc512258469)

[2.1.3 Intended Audience and Reading Suggestions 4](#_Toc512258470)

[2.1.4 Product Scope 4](#_Toc512258471)

[2.2 Overall Description 4](#_Toc512258472)

[2.2.1 Product Perspective 4](#_Toc512258473)

[2.2.2 Product Functions 4](#_Toc512258474)

[2.2.3 User Classes and Characteristics 5](#_Toc512258475)

[2.2.4 Operating Environment 5](#_Toc512258476)

[2.2.5 Design and Implementation Constraints 5](#_Toc512258477)

[2.2.6 User Documentation 5](#_Toc512258478)

[2.2.7 Assumptions and Dependencies 6](#_Toc512258479)

[2.3 External Interface Requirements 7](#_Toc512258480)

[2.3.1 User Interfaces 7](#_Toc512258481)

[2.3.2 Hardware Interfaces 8](#_Toc512258482)

[2.3.3 Software Interfaces 8](#_Toc512258483)

[2.3.4 Communications Interfaces 8](#_Toc512258484)

[2.4 System Features 9](#_Toc512258485)

[2.4.1 Temperature Reading 9](#_Toc512258486)

[2.4.2 Sound Reading 10](#_Toc512258487)

[2.4.3 Moisture Reading 11](#_Toc512258488)

[2.5 Other Nonfunctional Requirements 12](#_Toc512258489)

[2.5.1 Performance Requirements 12](#_Toc512258490)

[2.5.2 Safety Requirements 12](#_Toc512258491)

[2.5.3 Security Requirements 12](#_Toc512258492)

[2.5.4 Software Quality Attributes 12](#_Toc512258493)

[2.5.5 Business Rules 13](#_Toc512258494)

[3. Build Instructions 15](#_Toc512258495)

[3.4 Android Application 33](#_Toc512258496)

[3.5 Website 36](#_Toc512258497)

[4.1 Conclusion 38](#_Toc512258498)

[Bibliography 39](#_Toc512258499)

[Appendix A: Glossary 43](#_Toc512258500)

[Appendix B: Integrated Sensor Code 44](#_Toc512258501)

# List of Figures

[Figure 1 Android application concept GUI pictures. Consists of the initial design for the Login, Power Management, Register, Sound, and Status activity pages. 7](#_Toc510449163)

# Introduction

The HVAC monitoring system solves the technical problem of a more eco-friendly heating and cooling of a building. Other Smart HVAC systems do monitor temperature and moisture levels in the room but the noise generated from the HVAC systems is not monitored. The noise generating from HVAC systems can make it difficult for room occupants to hear, which can be a safety hazard as the listening to prolonged noise can damaged the occupants’ hearing. The noise can also make it difficult for occupants to hear noise generated from other systems (for example, an alarm system).

This IoT system will monitor and control the temperature, moisture and sound levels in the room using a Broadcom-based development platform (also known as the Raspberry Pi 3 [RPI]), temperature, moisture, and sound sensors, and will transmit the readings from the sensors to a shared hosted server (hostmonster.com) that uses a MySQL database with PHP (MyPHPAdmin). The database will interface with an Android application and a website for easy remote access for the user.

The report will detail the process for the concept, building, and feasibility of the project for other users to develop and improve upon the project. Instructions on where to find and how to build the HVAC system are included along with information on issues that we have encountered while working on the project.

# Project Description

# 2.1 Project Requirements Specification

## 2.1.1 Purpose

The purpose of this document is to detail the requirements for the HVAC Android application, and the website component, and the MySQL database component. The Android application’s purpose is to generate a user-friendly graphical interface for our HVAC hardware and to provide a status report containing the temperature readings, sound-based detection functionality of the system with the maintenance report and scheduler. The database’s purpose is to collect readings from the HVAC system for the Android application and website to fetch from.

## 2.1.2 Document Conventions

This document is written in English following software requirements specifications standards and occasionally uses bullet formatting as well as some concept images of the software product. Software development requirements featured in this document are listed with their own priority and weight.

## 2.1.3 Intended Audience and Reading Suggestions

The intended audience for this document is the project supervisors and application users. This document assumes that the audience has basic technical knowledge with Internet of Things (IoT) projects, Android mobile programming, WordPress/web programming, and MySQL/PHP.

## 2.1.4 Product Scope

The Android application scope is to help HVAC users to connect to, control and monitor the system remotely, and receive notifications about the system condition on the screen.

The database is there to store and retrieve information of the status of the hardware for every user-specified time interval.

# 2.2 Overall Description

## 2.2.1 Product Perspective

The product is the latest in the family of smart HVAC products, which monitor temperature and moisture. While maintaining traditional functionality it implements new technologies and provides new features and a new price.

## 2.2.2 Product Functions

* Read Temperature level
* Read Moisture level
* Determine System Functionality via Sound Sensor
* Store Status Reports in the database for the user to look back on

## 2.2.3 User Classes and Characteristics

This product will be used in residential, commercial and industrial buildings. The product will most frequently be used by homeowners and industrial workers.

## 2.2.4 Operating Environment

The operating environment for the project will be the Raspberry Pi 3 with Raspbian Linux, the Android mobile operating system, MySQL & PHP for the database environment, and finally, WordPress/HTML for the website.

## 2.2.5 Design and Implementation Constraints

Hardware limitations in terms of the sound sensor sensitivity (cannot be used in an extremely noisy environment), needs embedded C, Internet access for the database & website, minimum of Android API 5.0 (Lollipop), app encryption/security, hardware susceptible to extreme weather.

## 2.2.6 User Documentation

The technical report will be provided and the documentation and code will be available online. Main HVAC GitHub website: <https://github.com/fntj0052/HVAC>

Android Application GitHub website: <https://github.com/fntj0052/HVACApp>

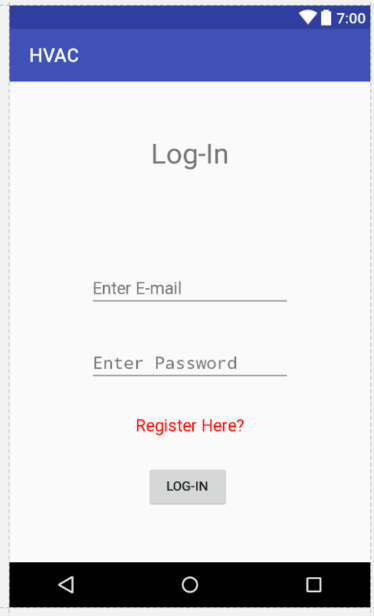
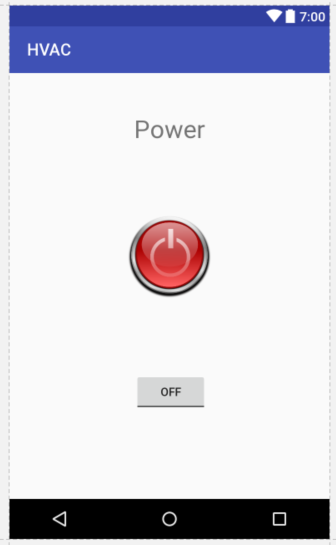
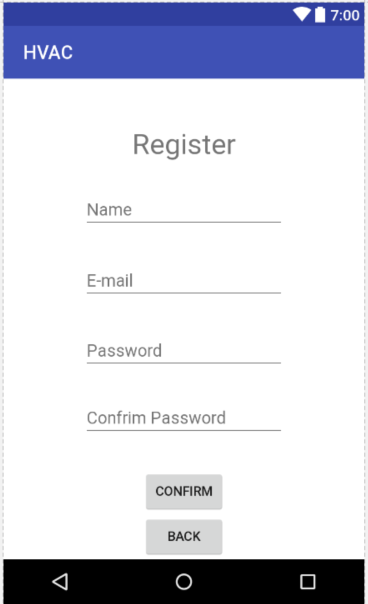
## 2.2.7 Assumptions and Dependencies

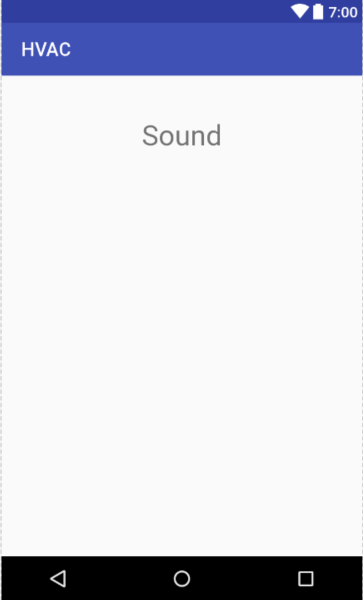
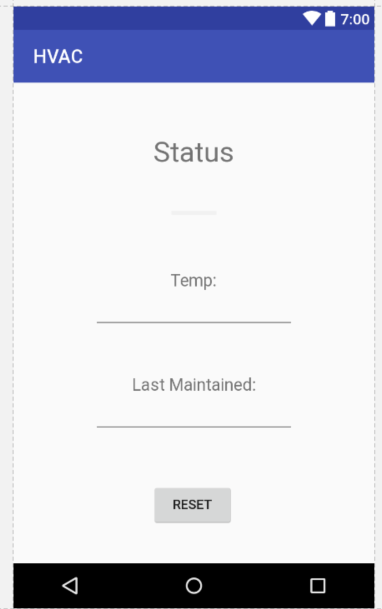
Updates and changes to the hosed server software and policies for MySQL and PHP, GitHub, and Android may affect the functionality of the product.

# 2.3 External Interface Requirements

## 2.3.1 User Interfaces

Android GUI standards will be implemented for the mobile application. The head developer of the Android application graphical user interface is Jan Fontanosa.



Figure 1 Android application concept GUI pictures. Consists of the initial design for the Login, Power Management, Register, Sound, and Status activity pages.

## 2.3.2 Hardware Interfaces

For the mobile application, any Android device running Android 5.0 or up will be able to use the application. The software on the Raspberry Pi 3 can be used through the LCD and physical buttons on the keypad. All project developers will be working on the project hardware.

## 2.3.3 Software Interfaces

For the web interface, a connection to the website through a browser will be necessary. The database can be viewed through the hosted server website using the account created on the website. The operating system used will be a Raspbian Linux. The HVAC system database will be developed and maintained by Maasha Maheson. The product will rely on analogRead to get the analog data and will be processed on C to display to the user through text/images.

## 2.3.4 Communications Interfaces

The product website can be accessed through any modern browser that uses the standard HTTP protocol. The website will be protected through free WordPress protection system but may be vulnerable to security issues. The project website will be developed by Vyacheslav Perepelytsya.

# 2.4 System Features

## 2.4.1 Temperature Reading

2.4.1.1 Description and Priority

The temperature reading is an essential component of the HVAC which will provide a reading of the room temperature to the user and determine whether the HVAC system needs to turn on or off. It’s of HIGH priority. The lead developer of the temperature reading sensor will be Maasha Maheson.

2.4.1.2 Stimulus/Response Sequences

The user is entitled to read the temperature as well as stop and start or restart the system.

2.4.1.3 Functional Requirements

The temperature sensor needs to be functioning as well as the Raspberry PI and the database. In case of a specific error there will be a specific error message to the user, in case of a power outage error or a system shutdown the user will experience a blank screen.

REQ-1: TEMP

## 2.4.2 Sound Reading

2.4.2.1 Description and Priority

The sound reading is MEDIUM priority component of the HVAC which will provide a reading of the noise level of the system environment to the user and determine if the HVAC system is operating at a safe sound level. The lead developer of the sound reading sensor will be Vyacheslav Perepelytsya.

2.4.2.2 Stimulus/Response Sequences

The user is entitled to read the sound level status as well as stop and start or restart the system.

2.4.2.3 Functional Requirements

The sound sensor needs to be functioning as well as the Raspberry PI and the database. In case of a specific error there will be a specific error message to the user, in case of a power outage error or a system shutdown the user will experience a blank screen.

REQ-1: SOUND

## 2.4.3 Moisture Reading

2.4.3.1 Description and Priority

The moisture reading is MEDIUM priority component of the HVAC which will provide a reading of the room moisture levels to the user and determine whether the HVAC system needs to turn on or off. The lead developer of the moisture reading sensor will be Jan Fontanosa.

2.4.3.2 Stimulus/Response Sequences

The user is entitled to read the moisture sensor status as well as stop and start or restart the system.

2.4.3.3 Functional Requirements

The moisture sensor needs to be functioning as well as the Raspberry PI and the database. In case of a specific error there will be a specific error message to the user, in case of a power outage error or a system shutdown the user will experience a blank screen.

REQ-1: MOIST

# 2.5 Other Nonfunctional Requirements

## 2.5.1 Performance Requirements

The Android device should be capable of running Android 5.0 (Lolipop) and have a good internet connection for the database to transfer information properly.

## 2.5.2 Safety Requirements

If the device is installed in an extreme weather environment it could damage or harm equipment and people. For example in an earthquake situation the device may affect or damage nearby equipment or people if it is installed improperly or unsafely. Floods and fires may also have unpredictable consequences for the system and nearby devices/people.

## 2.5.3 Security Requirements

The user does not need to provide any personal information aside from the email and password for the Android application.

## 2.5.4 Software Quality Attributes

Sensor accuracy, reliability and a user-friendly GUI are the main additional quality characteristics.

## 2.5.5 Business Rules

The owner of the product and the people they share the device access with will be the only users of the product and as such perform all the possible user roles. In cases of malfunction an additional person may be involved for the repairs.

# 3. Build Instructions

The Build Instructions section describes items purchased, the steps taken and any necessary modifications made during the construction of the smart HVAC monitoring system.

3.1 Bill of Materials

This subsection lists the parts needed to build the HVAC system:

* Raspberry Pi 3 – The Raspberry Pi 2 can be used but an external USB Wi-Fi adapter would need to be used with the Pi in order to maintain the HVAC system’s portability. A USB keyboard and mouse will be needed for the initial setup of the Pi. We have used a Raspberry Pi 3 kit similar to this: <https://www.amazon.ca/CanaKit-Raspberry-Micro-Supply-Listed/dp/B01E4HDIO4/ref=sr_1_9/144-7110608-6031153?ie=UTF8&qid=1520631689&sr=8-9&keywords=raspberry+pi+3>
* LCD Touchscreen Display – Any touchscreen can be used but we have chosen to use the Kuman 7” LCD Display due to its driver support of the Raspberry Pi 3. The screen is not currently sold on Amazon anymore but here is an alternative display that can be used: <https://www.amazon.ca/Kuman-Resistive-800x480-Display-Raspberry/dp/B01F4RSIA2/ref=sr_1_4?m=A3IRH1M32QHQ71&s=merchant-items&ie=UTF8&qid=1520632138&sr=1-4&keywords=raspberry+pi+touch+screen>
* Touch Sensor – This is used as an emergency power off switch for the smart HVAC monitoring system: <https://www.sainsmart.com/products/ttp223b-digital-touch-sensor>
* Moisture Sensor – This is used to monitor the moisture levels in the HVAC system: <https://www.sainsmart.com/products/water-sensor-with-free-cables>
* Sound Sensor – This is used to monitor the sound levels generated from the HVAC system. We have used two sound sensors for the HVAC system but the system can be calibrated to work with one sound sensor also: <https://www.sunfounder.com/sound-sensor-module.html>
* Temperature Sensor – This sensor is used to monitor the temperature levels of the system. We have decided to use the BME280 supplied for the Humber Student Sense Hat mentioned in the Sense Hat subsection. Here is the sensor from the manufacturer’s website: <https://www.sparkfun.com/products/13676>

3.2 Hardware Setup

This section details the setup of the hardware components of the HVAC monitoring system project. It consists of a Broadcom development platform (also known as the Raspberry Pi 3), multiple sensors (such as the temperature sensor, moisture sensor, sound level detector sensor, and the touch sensor), along with a touchscreen LCD interface. The smart HVAC monitoring system was originally written in Python code for the touch and moisture sensors by Jan Fontanosa but converted to Embedded C for better integration of sensor codes. The lead developer for the sound sensors is Vyacheslav Perepelytsya and the lead developer for the temperature sensor is Maasha Maheson.

3.2.1 Raspberry Pi

For this project, we have set up the Raspberry Pi according to the instructions hosted on the supplier’s website: we have used CanaKit’s *Quick Start Guide*, which is listed in the bibliography (CanaKit Corportation). We have also used Raspbian Stretch (Raspbian 9) for operating system as it has support for the libraries needed to use the smart HVAC monitoring system. This operating system and the software needed to operate the Sense Hat were configured according to the instructions listed on the six0four *Student Sense Hat* GitHub repository (Medri & Tian, 2017).

3.2.1.1 Raspberry Pi Setup & Operating System Installation

First, connect the HDMI cable to the appropriate connection ports on the Pi and the touchscreen LCD, or connect the cable to an HDMI port on any other display for initial setup. Then, connect an USB keyboard and mouse to the Pi, and optionally, the Ethernet cable to the correct ports on the Raspberry Pi. The Ethernet cable can be used for initial setup with the Pi 3 and is necessary for the Pi 2, if it does not have an external Wi-Fi adapter. Since we have used the Raspberry Pi 3, we have used the onboard Wi-Fi instead to connect to an open wireless network in order to download and install the root certificates needed to connect to an enterprise Wi-Fi network. This process is detailed later in the subsection titled *Enterprise Wi-Fi*. Afterwards, insert the microSD card preloaded with Raspbian Stretch (or NOOBS operating system, depending on what the supplier has included in their kit) and connect the Pi to the microUSB power charger to power it on. Once Raspbian has loaded up the desktop, install the following programs and enable the correct protocols detailed on the six0four *Student Sense Hat* GitHub instructions for the *Humber Parts Crib Raspberry Pi Image Creation*: <https://github.com/six0four/StudentSenseHat/blob/master/cribpisdcard.md>

3.2.1.2 Touchscreen LCD

First, connect the HDMI cable to the appropriate connection ports on the Pi and LCD. Then connect the microUSB to USB cable to the touch port on the LCD and the USB port on the Pi. Afterwards, go to the LCD screen manufacturer’s website and follow the documentation provide by the manufacturer to install the drivers for the screen to the Pi operating system that you are using. For our project, we used a Kuman 7 inch touchscreen LCD and thus we used their website and downloaded and installed the drivers as instructed (Kuman Ltd., 2016). We had to select the SC7B link in order to access the specific instructions for the chosen display. We had to modify the installation script included with the manufacturer’s drivers to use the root partition we were actually using for the Raspberry Pi 3. Without this modification, the Raspberry Pi would not boot correctly.

3.2.1.3 Enterprise Wi-Fi

In order to connect to an enterprise Wi-Fi network, we had to download and install the root certificate hosted by the network administrators. Once the root certificate was downloaded, we had to change the file type from .cer to .crt and move the certificate to the /usr/share/ca-certificates directory. Then we executed sudo dpkg-reconfigure ca-certificates in order to install the new certificate. Afterwards, we had to edit the configuration file for the wpa\_supplicant network program /etc/wpa\_supplicant/wpa\_supplicant.conf in order to configure the enterprise Wi-Fi as followed on the *Student Sense Hat* GitHub (Medri & Tian, 2017) and listed below:

network={

ssid="myWi-Fi@Humber"

priority=999

proto=RSN

key\_mgmt=WPA-EAP

pairwise=CCMP

auth\_alg=OPEN

eap=PEAP

identity="STUDENT ID"

password="PASSWORD"

phase1="peaplabel=0"

phase2="auth=MSCHAPV2"

}

3.2.2 Sensor Hat PCB

We have originally used the six0four *Student Sense Hat* GitHub instructions to build and configure the PCB and sensors and did not use a surface resistor. Originally, Vyacheslav Perepelytsya had stripped down the PCB by removing the transistors and resistors in order to provide proper voltages needed for the sound sensors to correctly work. The main difference is that the connection to the PCF is now direct, changing the voltage supply from 5V to 3.3V that is better suited for the project. Resistors R4-R7 were removed and instead, direct paths are provided through soldering cables and re-design. Q1 and Q2 (transistors) components are also unnecessary. After the sensor hat had been made, it can be mounted on top of the Raspberry Pi. This worked for the sound sensors but the problem with this first version of the PCB became apparent that it didn’t have any proper dedicated slots for the other new sensors: the dedicated temperature sensor slot had the wrong voltage supply for its SCL/SDA (5v) and the PCB has started showing signs of wear and tear when it has started malfunctioning because it got bent at the base, which is connected to the RPi. As a result, a second PCB was made with the idea of using the original six0four *Student Sense Hat* design with the resistors and transistors. This PCB had dedicated slots for the sound, moisture and touch sensors but it turned out that the design for this new PCB was erroneous as the copper lanes were on top of the PCB and were conflicting with the plastic slots. Soldering the components was difficult, if not impossible. A third PCB was made with these two concepts in mind: everything was connected to the 3.3 V trace as we realized that all the sensors are compatible with it. This meant that the resistors and transistors were removed as in the original design, and most of the traces were below the PCB for easier soldering. The traces above were the necessary GPIO slots for the moisture and touch sensor that could not go under the PCB as the pins were in the top row and soldering below was impossible due to the proximity of the plastic holding the pins. The other trace that runs mostly on top of the PCB was the ground trace as it was decided that it would be a good idea to separate it from the other lanes to remove possibilities of a shorted circuit. The final PCB also has dedicated slots for all the sensors with the appropriate amount of pins. A final innovation is that it has a relatively large area in the middle of the PCB, which is empty of traces and slots, to make sure that there is no interference of the electronic paths/devices.

3.2.2.1 Temperature Sensor

Maasha Maheson handled the integration of the temperature sensor to the smart HCAV monitoring system. For the system, we have used the temperature sensor provided in the Sense Hat kit – the BME280 by Sparkfun - and have followed the instructions for setting up the sensor according to the six0four *Student Sense Hat* GitHub page (Medri & Tian, 2017). To set up the temperature sensor to the Sense Hat PCB, we have connected the 3.3 V VCC connector on the BME280 sensor to Pin 1 on the Sense Hat GPIO pins, the ground connector to Pin 6 on the Pi, and the SCL and the SDA connectors to Pin 5 and 3 respectively. Then, we used the sample C library code for the BME280 sensor from Larry Bank’s GitHub repository to test the sensor (Bank, 2017). To create the sensor code for the smart HVAC monitoring system, we referred to and modified the bme280.c code from the *raspberry-pi-bme280* GitHub repository by Andrei Vainik to get a working sensor (Vainik, 2017).

3.2.2.2 Moisture Sensor

The moisture sensor used in the HVAC project was managed by Jan Fontanosa. The moisture sensor is used to check the moisture in an HVAC system the smart monitoring system is monitoring. It reads if the unit condition is wet or dry. When the system is dry according to the sensor, it is in good condition and does not need to have any user action but when the sensor detects any wetness, the HVAC monitoring system needs to be maintained. I have used the Water Sensor with Free Cables by Sainsmart (<https://www.sainsmart.com/products/water-sensor-with-free-cables>) for the Raspberry Pi 3. First, I connect the – (minus) pin to the ground pin on the Raspberry Pi 3 (Pin 6), the + (plus) pin to the 5 V on Pi (pin 2), and the S-indicated pin to pin 18 according to the Raspberry Pi 3 GPIO pin diagram [GPIO24]. I used a Raspberry Pi 3 pin diagram online to help with the connections: <https://pinout.xyz/>. Then, I connect the sensor to the Sense Hat, which should be connected to the Pi.

I originally wrote the code in Python for this sensor and the touch sensor in the previous semester for Hardware Production Technology, but I have converted the code to C to better integrate with the sound sensors Vyacheslav Perepelytsya has worked on and the temperature sensor Maasha Maheson has worked on. I had to use different Raspberry Pi GPIO libraries for the conversion but the pin locations were different in the Python RPi.GPIO library from the C wiringPi library. I have used GPIO 8 (pin 24) in Python and in C, pin 12. I had to make sure I used the right methods since GPIO.input() is used for Python while digitalRead() is used in C. Below is the original Python code, which included functionality for email notifications, and in Appendix B is the C code conversion in the combined sensor code:

########################################

# Login credentials to send email#

########################################

username = 'jv2ee@yahoo.com'

password = 'EmailPassword'

############################

# General Email Parameters #

############################

From = "jv2ee@yahoo.com"

To = "jv2ee@yahoo.com"

#######################################

# Email Parameters when sensor is Wet #

#######################################

Subject\_wet = "RPi Water Sensor is WET"

Body\_wet = "Your water sensor is wet."

#######################################

# Email Parameters when semsor is Dry #

#######################################

Subject\_dry = "RPi Water Sensor is DRY"

Body\_dry = " Your water sensor is dry again!"

import smtplib

from email.mime.text import MIMEText

import RPi.GPIO as GPIO

import string

import time

#Function Definitions

#takes either "wet" or "dry" as the condition.

def email(condition):

print "Attempting to send email"

if condition == 'wet':

Body = string.join((

"From: %s" % From,

"To: %s" % To,

"Subject: %s" % Subject\_wet,

"",

Body\_wet,

), "\r\n")

if condition == 'dry':

Body = string.join((

"From: %s" % From,

"To: %s" % To,

"Subject: %s" % Subject\_dry,

"",

Body\_dry,

), "\r\n")

# The actual mail send

server = smtplib.SMTP('smtp.gmail.com:587')

server.starttls()

print "Logging in..."

server.login(username,password)

print "Logged in as "+username+"."

server.sendmail(From, [To], Body)

server.quit()

print "Email sent."

#Tests whether wter is present.

# returns 0 for dry

# returns 1 for wet

# tested to work on pin 18

def RCtime (RCpin):

reading = 0

GPIO.setmode(GPIO.BCM)

GPIO.setup(RCpin, GPIO.OUT)

GPIO.output(RCpin, GPIO.LOW)

time.sleep(0.1)

GPIO.setup(RCpin, GPIO.IN)

# This takes about 1 millisecond per loop cycle

while True:

if (GPIO.input(RCpin) == GPIO.LOW):

reading += 1

if reading >= 1000:

return 0

if (GPIO.input(RCpin) != GPIO.LOW):

return 1

def buzz\_on (pin):

GPIO.setmode(GPIO.BCM)

GPIO.setup(pin, GPIO.OUT)

GPIO.output(pin, GPIO.HIGH)

def buzz\_off(pin):

GPIO.setmode(GPIO.BCM)

GPIO.setup(pin, GPIO.OUT)

GPIO.output(pin, GPIO.LOW)

# Main Loop

print 'Waiting for wetness...'

while True:

time.sleep(1)

if RCtime(18) == 1:

print "Sensor is wet"

email('wet')

while True:

time.sleep(1)

if RCtime(18) == 1:

print "Sensor is still wet..."

buzz\_on(17)

continue

if RCtime(18) == 0:

buzz\_off(17)

print "Sensor is dry again"

email('dry')

print "Waiting for wetness..."

break

I had an issue allocating the pins for power and ground from the moisture sensor to the touch sensor. The Power pin used 3.3V (pin 17) and Ground used pin 9 but in the previous semester, I used 5V for the power pin, which I have changed to 3.3V in the final PCB because of the way the PCB is designed. In the final PCB version, we needed to change the original Sense Hat PCB schematic to one that worked for all of the sensors so that the sensors would have enough power to work.

* + - 1. Sound Level Detector Sensor

Sound sensors were connected each to a 3.3V voltage output and ground on the PCB, as well as 1 PCF input slot for each, all of this was connected to the modified PCB (See modifications under PCB heading) from CENG Hardware Project. The time needed was no more than 2 hours of soldering, no more than 2 hours of programming and calibration, and was around 30 minutes of connecting parts.

The program needed to test the sound sensor is included below and in the HVAC GitHub link under the name NoiseDetector.c. Run the code with administrator permissions: sudo ./NoiseDetector. Compile the code with gcc –Wall –o NoiseDetector NoiseDetector.c –lwiringpi -lm

Each of the sound sensors should be tested and calibrated appropriately to have close to equal sound sensitivity. If not, the PCF pin slots or the sound sensor hardware itself may need to be replaced. Changing cables and the PCF input slots for the sound sensors can provide a different result.

#include <stdio.h>  
#include <stdlib.h>  
#include <wiringPi.h>  
#include <pcf8591.h>  
#include <math.h>

#define PCF 120

int main (void) {   
int value1, value2;   
int data1, data2;   
int counter = 0;   
int step = 1;   
//1 int i = 0;  
 int offset = 85;   
//85 int deviation = 10;   
//10 int noiseCounter = 0;  
int noiseDuration = 5;   
//20 int noiseOffCounter = 0;  
 int noiseOffDuration = 1000;  
 //1000 int noise = 0;

if (wiringPiSetup () == -1) {  
 printf("Error at wiringPiSetup()");  
 return 1;  
 }

pinMode (0, OUTPUT) ;   
// aka BCM\_GPIO pin 17 pinMode (1, OUTPUT) ;   
// aka BCM\_GPIO pin 18

// Setup pcf8591 on base pin 120, and address 0x48   
// printf("%d\n", ++step);

if (pcf8591Setup (PCF, 0x48) == -1) {  
 printf("Error at pcf8591Setup()");  
 return 1 ;  
 }  
 printf("Lowest noise level \t\t\t%d\n", offset);  
 printf("Data has been generalized with weight value of %d\n", step);

while(1) // loop forever   
{ data1 += analogRead (PCF + 0);  
 //white - local data2 += analogRead (PCF + 1);  
 //yellow - remote counter++;  
 if(noise) { noiseOffCounter++;  
 }

if(counter == step ) {   
counter = 0;  
 value1 = round((double)data1 / step);  
 value2 = round((double)data2 / step);  
 data1 = 0;  
 data2 = 0;

if ((value1 < offset) || (value2 < offset)) {

if( abs (value1 - value2) < deviation) {  
 if (++noiseCounter > noiseDuration) {  
 noiseCounter = 0; noiseOffCounter = 0; noise = 1;   
} } else {  
 if(noise) {  
 if(noiseOffCounter > noiseOffDuration) {  
 noiseOffCounter = 0; noise = 0;   
} } else if( --noiseCounter < 0) {  
 noiseCounter = 0;   
} }

if(noise) {  
 printf("\n\tNOISE !!!! #1:%3d #2:%3d", value1, value2);  
 //both microphones are sensing noise - red digitalWrite (0, LOW);  
 digitalWrite (1, HIGH);  
 } else {   
printf("\n\tVoice In!! #1:%3d #2:%3d", value1, value2);   
// Voice in at any of microphones - green digitalWrite (0, HIGH); digitalWrite (1, LOW); }

for(i = offset; (i > offset - value1) && (i > 0); i--) {  
 printf(" ");   
} for(i = offset; i > value1; i--) {  
 printf("-");  
 }   
printf("|");  
 for(i = offset; i > value2; i--) {  
 printf("+");  
 } } else {  
 if(!noise) { digitalWrite (0, LOW);  
 digitalWrite (1, LOW); } } } }

return 0; }

3.2.2.4 Touch Sensor

The touch sensor aspect of the HVAC project was managed by Jan Fontanosa. The touch sensor is used as a killswitch in the monitoring system so that the system can be turned off quickly if there was an emergency situation with the system. In the previous semester for Hardware Production Technology, I used a touch sensor by Sainsmart (TTP223B Digital Touch Sensor) and used Python to code the sensor. In this semester for Computer Systems Project, I have converted the sensor code to C for better integration with the other sensors. I have included the original Python code below and the integrated C code in Appendix B.

import time

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

padPin = 16

GPIO.setup(padPin, GPIO.IN)

alreadyPressed = False

while True:

padPressed = GPIO.input(padPin)

if padPressed and not alreadyPressed:

print "pressed"

alreadyPressed = padPressed

time.sleep(0.1)

To connect the sensor to the Raspberry Pi 3, I connected the G pin to the ground pin (GPIO Pin 9), the V pin to the 3.3 V pin (GPIO pin 1), and the out pin to GPIO4 (pin 16). I used a Raspberry Pi 3 pin diagram online to help with the connections: <https://pinout.xyz/>.

These are the issues I have encountered with working with the touch sensor: first, the pin locations were different in Python and C because of the different libraries I used. For example, I used the RPi.GPIO library in Python – this would use GPIO 23 for pin 16 – while in C code, I used the wiringPi library, which would use GPIO 4 for pin 16. According to the wiringPi library website, it uses a wiring system similar to the Arduino (<http://wiringpi.com/>) so I had to troubleshoot the touch sensor by checking which pin the wiringPi library would need to use. I also needed to double-check the methods used in the library since they are different: GPIO.input() for Python versus digitalRead() in the wiringPi C library. Another issue I faced was that the pins for the sensor needed to connect to all of the other sensors for the HVAC project in the final version of the Sense Hat PCB: we needed to make sure that all of the sensors had the right amount of power to work as the touch sensor needs 3.3V while the other sensors are good with 5V. Lastly, I was having issues communicating with the touch sensors using the Android application without using the database. I could not find a way to implement the killswitch in the Android application without needing to create or use an existing terminal/SSH application and then using SSH to connect to the Raspberry Pi 3. ([Raspberry Pi Foundation,](file:///C:\Users\n01133953\Downloads\Raspberry%20Pi%20Foundation,%20https:\www.raspberrypi.org\documentation\remote-access\ssh\android.md) ) I could use this method to cancel the HVAC program on the RPI or use it to shut down the RPI but this could be a security issue.

3.3 Database

Maasha Maheson was in charge of the database integration. We originally went with the option of using Amazon IoT with DynamoDB for our database needs due to the ability of DynamoDB to handle large sets of data because of its use of NoSQL. We switched from this option because we had issues configuring the database with our Raspberry Pi 3 and sensors. We ultimately decided to use a shared hosted server (HostMonster.com, 2015) that uses a MySQL database. This was the next best solution for the project because we only had to query the database a relatively small amount of times during the time of the project and we were able to negotiate a short-term contract with the server renters. Ideally, we would have used Amazon IoT and DynamoDB so we have listed the steps we have taken with the Amazon services along with the hosted server below.

First, we had signing up for an Amazon AWS free-tier AWS account since the read and write capacities for DynamoDB offered with the free-tier would have been sufficient for our needs. After signing up the free-tier AWS account, we clicked on the IoT services button and we followed the *Connecting to the Raspberry Pi* section in the AWS IoT API guide to configure the IoT services on the Pi (Amazon Web Services, Inc., n.d.). We also consulted the first part of the *Connecting Raspberry Pi Zero with Amazon Web Services IoT* guide by goMake using the appropriate substitutions from the Raspberry Pi Zero to the Raspberry Pi 3 (goMake, 2017). We then followed the Embedded C SDK guide to install the SDK onto Raspberry Pi: we added the source code for the external programs into the appropriate folders and then executed the make command on the SDK parent directory to compile both the external dependencies for the SDK and the actual Embedded C SDK (Amazon Web Services, Inc., n.d.). We had some issues using make which was suggested when troubleshooting the SDK installation as Raspbian Stretch - the Raspberry Pi Linux-based operating system we have chosen to use for this smart HVAC monitoring system – at the time of writing did not have all of the packages needed in the Makefile so we had to use cmake instead (Amazon Web Services, Inc., n.d.). To test AWS IoT, we have used the sample programs in folder to change the config.h file for the program to connect to the Rest API link listed in the AWS IoT website (Amazon Web Services, Inc., n.d.).

To setup DynamoDB, we have used the created AWS account to create a DynamoDB table with the default options configured as these were the recommended options for users of the free-tier account (Amazon Web Services, Inc., 2017). As a precaution, the default options for read and write capacities do cost money (we would have been charged the relatively small amount of about fifty cents per month). We then had to name the table and set up the column names, values, and keys. For the HVAC monitoring system, we decided to make the user email our primary key and have the sensor data – each listed separately in their own column with the appropriate integer value assigned to them – along with a timestamp as the subsequent columns.

At this point, we had encountered issues with having the database connect to the Raspberry Pi 3 and switched to the MySQL and MyPHPAdmin shared hosted server database. First, we had created an account with the server we had chosen to use at HostMonster and signed into the file manager to create a separate folder where we can create the PHP files – the PHP code used for the project are placed in the Database folder in the HVAC repository: <https://github.com/fntj0052/HVAC/tree/master/Database> – and change the permissions so that the PHP can interface with the integrated sensor code on the Raspberry Pi 3 (HostMonster.com). One issue we encountered was that we had to make sure that the PHP code was using appropriate methods to retrieve and send data to the Raspberry Pi 3 and the Android application, respectively. We used the get method for the PHP so that the data could be sent to the sensor database whereas we used the post method for the Android application as sensitive data would be sent to the user account information database.

We then created a separate user with their own password so that the MySQL database that we were going to use can be accessed from the Raspberry Pi 3 through the integrated sensor code without potentially affecting the entire server. Afterwards, we logged into the MyPHPAdmin function on the server and created a MySQL database to hold the sensor data with *hvac* as the table name, and the id, email, sound, moisture, temp, and timestamp as the column names. We assigned the each of the columns their appropriate value types and keys: id was used for autoincrementing, email was a text type and the primary key of the database, timestamp was the current timestamp, and the other columns took integer values. For the second database, we used it to hold the email and password information of the users that signed up for the account using the Android application. We called this database hvacemail, and assigned the column names of id, email, password, and timestamp with appropriate values for each column. Email remained the primary key for this database.

# 3.4 Android Application

Jan Fontanosa was in charge of the Android application. The application allows the user to register an account with the project, login to this account, monitor the HVAC system, and notify the user of the HVAC condition. I have used Android Studio to create the application, and I have used the Android Studio built-in emulator to test it. The Android application can be designed in anyway provided that the main functionality is there. The main Java and XML code is included on the GitHub repository for the Android application: <https://github.com/fntj0052/HVACApp>

I had some issues with creating the Android application: I built a new application from scratch since I was not continuing the application from semester 5 as all project members were working on different projects in the previous semester. This application has a basic user-friendly interface due to this shortened time. I also had some issues writing to the email database Maasha Maheson has created as I had used the POST method for the Android application instead of the GET method. This was mentioned in the database section of the report. The last problem that occurred was I had to figure out what output was needed for the temperature and the sound activities.   
To create the Android application, I set up a new project in Android Studio with the minimum of API 16 (Android 4.1 – Jelly Bean) and created an empty activity. I also created a Manifest file with the Internet permission added to the file <uses-permission android:name=”android.permission.INTERNET” /> so the application can access the database and the URL for the GitHub website. I used RelativeLayout for the default layout of the application since it is an easier layout to work with and used LinearLayout for HomePage.java with orientation=”vertical” so that it would look even and neat.

The first activity the user would be greeted with is HomePage.java which has all of the activities on the application. The activity has imagebuttons that redirects the user to the corresponding activity through an intent, which needs an onClick statement. I used this icon generator to create multiple sized images for the imagebuttons and the application: <https://romannurik.github.io/AndroidAssetStudio/icons-generic.html#source.type=clipart&source.clipart=ac_unit&source.space.trim=1&source.space.pad=0&size=32&padding=8&color=rgba(0%2C%200%2C%200%2C%200.54)&name=ic_ac_unit>

The Oncreate option menu is in every activity and the menu page shortcuts redirects to the intent selected by the user. I used switch case in order to implement this with each activity in the application. I have included a link to the project website through the about option menu. I had to delete an entire java class, power.class, which would supposedly activate the killswitch using the touch sensor when the toggle button is pressed.

The LogInActivity.java activity prompt the user to input their new or already registered email, add the email and password into the designated editText field, and uses a set of rules to validate the input. For email validation, you can use “android.util.Patterns.EMAIL\_ADDRESS.matcher” if the user inputted an invalid field or leaves it empty. It will grab from the database and prompt a message “Please enter a valid email address”. For password validation, user must need to input the exact password created on the Registration.Activity because the Login.Activity will POST from the Database itself. If the user inputs the wrong password, the user will prompted with a message “Incorrect Password/Nothing typed in”. If the user inputted nothing in the editText field, it will lead to a failed login and restricts the user from continue to the next activity/intent(Homepage.activity). Also this Page will re-direct the user if the user does not have an account yet by clicking on the red “Register Here?” text.

For the BackgroundWorker.java, it extends to the login.activity for getting the database using the POST method. It helps the connection to the database by requesting the connection = valid, which will get the output from the outputStream in the httpURLConnection. It uses “UTF-8” and “iso-8859-1” compatibility and sets the parameters for email= 1 and password=2. It connects to the database.

For the RegisterActivity.java, the user needs to the input personal information to all fields. It connects to the database using the POST method “class SendPostReqAsyncTask”. I had issues writing to the database due to the aforementioned POST versus GET method issue.

For the SoundActivity.java, it shows the user the sound level from the database, where 0 = “Normal Sound level” and 1 = “High Sound level! Check system”. It uses GET method for the database connection. The database strings are parsed using JSON and show the id sound and timestamp from the database.

For Result.java, it uses the String for id/ sound/ timestamp readings from database while ResultAdapter.java makes an Array that communicates to ArrayAdapter for parsing purposes and it gets the object from the database. For DisplayListview.java, JSON is parsed using a jsonObject and jsonArray. For StatsActivity.java, which shows the user the temperature of the unit and the last maintenance made/done using a hardcoded String EVENT (private String EVENT\_DATE\_TIME = “2018-04-27 12:30:00”) with a DATE\_FORMAT = “yyyy-MM-dd HH:mm:ss” for the custom progressbar. Lastly, I used the useLibrary “org.apache.http.legacy” for the build.gradle to connect to the database. The last issue I had was uploading it to GitHub using the built-in Android Studio feature as uploading it normally like I did for other code caused GitHub to not accept the files because there were too many.

# 3.5 Website

The website is lead developed by Vyacheslav Perepelytsya. Overall the website development was redone and rethought several times. The website interface was under development using HTML, PHP, CSS, HTTP RESTful protocol, Simba JDBC drivers, ReactJS and NodeJS 3 tier system to interface with the Amazon AWS IoT and DynamoDB database created before. At one point we have used Tomcat server for local testing of the website and Spring Tools Suite to develop the code. Originally the tool of choice was WordPress but it was not compatible with DynamoDB, however we then reverted back to WordPress for the personal website as it was decided that DynamoDB was not the optimal choice for our project in favor of SQL, which allows us to have PHP based web service to handle the database and the project website on the other hand would provide the link to install the application and other information. In other words the website side of the project was split in two parts, on the one hand there is the user friendly website <https://cenghvac.wordpress.com/> which was developed using WordPress, it consists of a home page introducing the user to the webpage, an about page with background information of the project, an application page to download the latest version of the application for the HVAC device and a contact page allowing the user to write comments to the developers using a form. On the other hand there is <http://justlikerav.com/hvac/> which is the website (Apache Server) used for the various web services for the Database portion of the project, which was developed mainly by Maasha Maheson (See Database portion of the technical report for more details).

# 4.1 Conclusion

This technical report has shown that an Internet of Things-based Heating, Ventilation, and Air Conditioning (HVAC) monitoring system is feasible to build using the build instructions and other documentation included in the report. As the system can be optimised according to their needs and that the parts for this system are relatively easy to obtain and cost-efficient to purchase, the smart HVAC monitoring system ensures that an HVAC system is energy-efficient. The HVAC monitoring system comprises from a Raspberry Pi 3 with sensors that check for the temperature, moisture and sound levels in the system, and an LCD touchscreen and touch sensor that is used to control the system. The system also has an Android application and web interface, which are used to interface with the hardware, and the database that contains the sensor data. All of these components create a self-sufficient HVAC monitoring system that makes it easy for users to maintain their residential, commercial and industrial HVAC systems.

# Bibliography

Al-Ali, A. R., Alikarar, M., Gupta, R., Rashid, M., Zualkernan, I.A. (2017). A smart home energy management system using IoT and big data analytics approach. *IEEE Transactions on Consumer Electronics, 63*(4), 426-434. Retrieved from <http://ieeexplore.ieee.org/document/8246800>

Amazon Web Services, Inc. (2012, August 10). Amazon DynamoDB Developer Guide. Retrieved from <http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/>

Amazon Web Services, Inc. (n.d). AWS IoT Developer Guide. Retrieved from <http://docs.aws.amazon.com/iot/latest/developerguide/>

Bank, Larry. (2017). bitbank2/bme280: A simple C library (Linux + Arduino) to read the calibrated temperature/pressure/humidity values from a Bosch BME280 I2C sensor. Retrieved from <https://github.com/bitbank2/bme280>

CanaKit Corporation. (n.d). Raspberry PI Quick Start Guide. Retrieved from <https://www.canakit.com/quick-start/pi>

goMake. (2017, August 3). Connecting Raspberry Pi Zero with Amazon Web Services IoT – Part I: Working with Amazon IoT. Retrieved from [https://medium.com/@gomaketeam/connecting-raspberry-pi-zero-with-amazon-web -services-iot-part-i-working-with-aws-iot-caa8a0f6f91b](https://medium.com/@gomaketeam/connecting-raspberry-pi-zero-with-amazon-web-services-iot-part-i-working-with-aws-iot-caa8a0f6f91b)

Google LLC & Open Handset Alliance. (n.d). Android API Guide. Retrieved from <https://developer.android.com/guide/index.html>

Ecovent Systems Inc. (n.d.). *Ecovent*. Retrieved from <https://www.ecoventsystems.com/smart>

HostMonster.com. (2015). *Professional Web Hosting from HostMonster*. Retrieved from <https://www.hostmonster.com/>

Kuman Ltd. (2016, May 12). Documents and recources,Support & Tutorials. Retrieved from <http://www.kumantech.com/help/documents-and-recources_h0037.html>

Institute of Electrical and Electronics Engineers. (2015, August 28). IEEE Xplore Digital Library [Online]. Retrieved from <https://ieeexplore.ieee.org/search/advsearch.jsp>

Medri, Kristian & Tian, Austin. (2017). six0four/StudentSenseHat: Student Sense Hat. Retrieved from https://github.com/six0four/StudentSenseHat

Raspberry Pi Foundation. (2017, March 8). SSH Using Android – Raspberry Pi Documentation. Retrieved from [https://www.raspberrypi.org/documentation/remote-access /ssh/android.md](https://www.raspberrypi.org/documentation/remote-access%09/ssh/android.md)

SoundFounder.com. (2017, April 13). Lesson 19 Sound Sensor. Retrieved from <https://www.sunfounder.com/learn/lesson-19-sound-sensor-sensor-kit-v2-0-for-b.html>

Vainik, Andrei. (2017). andreiva/raspberry-pi-bme280: Raspberry Pi BME280 driver. Retrieved from <https://github.com/andreiva/raspberry-pi-bme280>

Wordpress.org. (n.d). WordPress APIs << WordPress Codex. Retrieved from <https://codex.wordpress.org/WordPress_APIs>

# Appendix A: Glossary

**AWS** – Amazon Web Services. Cloud computing-based software.

**DynamoDB** – A NoSQL database service that can be used with AWS

**GitHub** – A website to share and develop code/projects using version control

**IoT** – Internet of Things. Products/systems that connect to and communicate using the Internet

**Raspberry PI 3** (RPI) – a recent version of a portable computer/Broadcom development platform used for practical projects and programming education.

# Appendix B: Integrated Sensor Code

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <wiringPi.h>

#include <pcf8591.h>

#include <stdint.h>

#include <time.h>

#include <math.h>

#include <wiringPiI2C.h>

#include "bme280.h"

#include <netinet/tcp.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <netdb.h>

#include <pthread.h>

#define BUFFER\_SIZE 1024

#define PCF 120

char request[512];

char buffer[BUFFER\_SIZE];

int fd;

int fd2;

bme280\_calib\_data cal;

bme280\_raw\_data raw;

float temperature;

int moistureread;

int RCpin = 5;

int dry = 0;

int wet = 1;

int Rpin = 0;

char email[50] = "hvacuser@gmail.com";

float temp;

float moisture;

short sound;

int result;

int noise = 0;

int socket\_connect(char \*host, in\_port\_t port){

struct hostent \*hp;

struct sockaddr\_in addr;

int on = 1, sock;

printf("%s, \n", host);

if((hp = gethostbyname(host)) == NULL){

herror("gethostbyname");

exit(1);

}

bcopy(hp->h\_addr, &addr.sin\_addr, hp->h\_length);

addr.sin\_port = htons(port);

addr.sin\_family = AF\_INET;

sock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

setsockopt(sock, IPPROTO\_TCP, TCP\_NODELAY, (const char \*)&on, sizeof(int));

if(sock == -1){

perror("setsockopt");

exit(1);

}

if(connect(sock, (struct sockaddr \*)&addr, sizeof(struct sockaddr\_in)) == -1){

perror("connect");

exit(1);

}

return sock;

}

int sendData(char \*email, float temp, float moisture, int noise){

//char request[] = "GET /hvacWS/sensors/read?from=now&to=now\r\n";

//char request[] = "GET /hvac/read.php HTTP/1.1\r\nHost: www.justlikerav.com:80\r\n\r\n";

char requestformat[] = "GET /hvac/datatransfer.php?email=%s&temp=%f&moisture=%f&sound=%d HTTP/1.1\r\nHost: www.justlikerav.com:80\r\n\r\n";

sprintf(request, requestformat, email, temp, moisture, noise);

write(fd2, request, strlen(request)); // write(fd, char[]\*, len);

/\*

bzero(buffer, BUFFER\_SIZE);

while(read(fd2, buffer, BUFFER\_SIZE - 1) != 0){

fprintf(stdout, "%s", buffer);

bzero(buffer, BUFFER\_SIZE);

}

\*/

//todo error analysation

return 0;

}

int bme280\_init() {

fd = wiringPiI2CSetup(BME280\_ADDRESS);

if(fd < 0) {

printf("Device not found");

exit(-1);

}

return 0;

}

int stream\_sensor\_data() {

readCalibrationData(fd, &cal);

wiringPiI2CWriteReg8(fd, 0xf2, 0x01); // humidity oversampling x 1

wiringPiI2CWriteReg8(fd, 0xf4, 0x25); // pressure and temperature oversampling x 1, mode normal

getRawData(fd, &raw);

int32\_t t\_fine = getTemperatureCalibration(&cal, raw.temperature);

temperature = compensateTemperature(t\_fine); // C

// float p = compensatePressure(raw.pressure, &cal, t\_fine) / 100; // hPa

// float h = compensateHumidity(raw.humidity, &cal, t\_fine); // %

// float a = getAltitude(p); // meters

/\*

printf("{\"sensor\":\"bme280\", \"humidity\":%.2f, \"pressure\":%.2f,"

" \"temperature\":%.2f, \"altitude\":%.2f, \"timestamp\":%d}\n",

h, p, t, a, (int)time(NULL));

\*/

return 0;

}

int32\_t getTemperatureCalibration(bme280\_calib\_data \*cal, int32\_t adc\_T) {

int32\_t var1 = ((((adc\_T>>3) - ((int32\_t)cal->dig\_T1 <<1))) \*

((int32\_t)cal->dig\_T2)) >> 11;

int32\_t var2 = (((((adc\_T>>4) - ((int32\_t)cal->dig\_T1)) \*

((adc\_T>>4) - ((int32\_t)cal->dig\_T1))) >> 12) \*

((int32\_t)cal->dig\_T3)) >> 14;

return var1 + var2;

}

void readCalibrationData(int fd, bme280\_calib\_data \*data) {

data->dig\_T1 = (uint16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_T1);

data->dig\_T2 = (int16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_T2);

data->dig\_T3 = (int16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_T3);

data->dig\_P1 = (uint16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_P1);

data->dig\_P2 = (int16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_P2);

data->dig\_P3 = (int16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_P3);

data->dig\_P4 = (int16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_P4);

data->dig\_P5 = (int16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_P5);

data->dig\_P6 = (int16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_P6);

data->dig\_P7 = (int16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_P7);

data->dig\_P8 = (int16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_P8);

data->dig\_P9 = (int16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_P9);

data->dig\_H1 = (uint8\_t)wiringPiI2CReadReg8(fd, BME280\_REGISTER\_DIG\_H1);

data->dig\_H2 = (int16\_t)wiringPiI2CReadReg16(fd, BME280\_REGISTER\_DIG\_H2);

data->dig\_H3 = (uint8\_t)wiringPiI2CReadReg8(fd, BME280\_REGISTER\_DIG\_H3);

data->dig\_H4 = (wiringPiI2CReadReg8(fd, BME280\_REGISTER\_DIG\_H4) << 4) | (wiringPiI2CReadReg8(fd, BME280\_REGISTER\_DIG\_H4+1) & 0xF);

data->dig\_H5 = (wiringPiI2CReadReg8(fd, BME280\_REGISTER\_DIG\_H5+1) << 4) | (wiringPiI2CReadReg8(fd, BME280\_REGISTER\_DIG\_H5) >> 4);

data->dig\_H6 = (int8\_t)wiringPiI2CReadReg8(fd, BME280\_REGISTER\_DIG\_H6);

}

float compensateTemperature(int32\_t t\_fine) {

float T = (t\_fine \* 5 + 128) >> 8;

return T/100;

}

float compensatePressure(int32\_t adc\_P, bme280\_calib\_data \*cal, int32\_t t\_fine) {

int64\_t var1, var2, p;

var1 = ((int64\_t)t\_fine) - 128000;

var2 = var1 \* var1 \* (int64\_t)cal->dig\_P6;

var2 = var2 + ((var1\*(int64\_t)cal->dig\_P5)<<17);

var2 = var2 + (((int64\_t)cal->dig\_P4)<<35);

var1 = ((var1 \* var1 \* (int64\_t)cal->dig\_P3)>>8) +

((var1 \* (int64\_t)cal->dig\_P2)<<12);

var1 = (((((int64\_t)1)<<47)+var1))\*((int64\_t)cal->dig\_P1)>>33;

if (var1 == 0) {

return 0; // avoid exception caused by division by zero

}

p = 1048576 - adc\_P;

p = (((p<<31) - var2)\*3125) / var1;

var1 = (((int64\_t)cal->dig\_P9) \* (p>>13) \* (p>>13)) >> 25;

var2 = (((int64\_t)cal->dig\_P8) \* p) >> 19;

p = ((p + var1 + var2) >> 8) + (((int64\_t)cal->dig\_P7)<<4);

return (float)p/256;

}

float compensateHumidity(int32\_t adc\_H, bme280\_calib\_data \*cal, int32\_t t\_fine) {

int32\_t v\_x1\_u32r;

v\_x1\_u32r = (t\_fine - ((int32\_t)76800));

v\_x1\_u32r = (((((adc\_H << 14) - (((int32\_t)cal->dig\_H4) << 20) -

(((int32\_t)cal->dig\_H5) \* v\_x1\_u32r)) + ((int32\_t)16384)) >> 15) \*

(((((((v\_x1\_u32r \* ((int32\_t)cal->dig\_H6)) >> 10) \*

(((v\_x1\_u32r \* ((int32\_t)cal->dig\_H3)) >> 11) + ((int32\_t)32768))) >> 10) +

((int32\_t)2097152)) \* ((int32\_t)cal->dig\_H2) + 8192) >> 14));

v\_x1\_u32r = (v\_x1\_u32r - (((((v\_x1\_u32r >> 15) \* (v\_x1\_u32r >> 15)) >> 7) \*

((int32\_t)cal->dig\_H1)) >> 4));

v\_x1\_u32r = (v\_x1\_u32r < 0) ? 0 : v\_x1\_u32r;

v\_x1\_u32r = (v\_x1\_u32r > 419430400) ? 419430400 : v\_x1\_u32r;

float h = (v\_x1\_u32r>>12);

return h / 1024.0;

}

void getRawData(int fd, bme280\_raw\_data \*raw) {

wiringPiI2CWrite(fd, 0xf7);

raw->pmsb = wiringPiI2CRead(fd);

raw->plsb = wiringPiI2CRead(fd);

raw->pxsb = wiringPiI2CRead(fd);

raw->tmsb = wiringPiI2CRead(fd);

raw->tlsb = wiringPiI2CRead(fd);

raw->txsb = wiringPiI2CRead(fd);

raw->hmsb = wiringPiI2CRead(fd);

raw->hlsb = wiringPiI2CRead(fd);

raw->temperature = 0;

raw->temperature = (raw->temperature | raw->tmsb) << 8;

raw->temperature = (raw->temperature | raw->tlsb) << 8;

raw->temperature = (raw->temperature | raw->txsb) >> 4;

raw->pressure = 0;

raw->pressure = (raw->pressure | raw->pmsb) << 8;

raw->pressure = (raw->pressure | raw->plsb) << 8;

raw->pressure = (raw->pressure | raw->pxsb) >> 4;

raw->humidity = 0;

raw->humidity = (raw->humidity | raw->hmsb) << 8;

raw->humidity = (raw->humidity | raw->hlsb);

}

float getAltitude(float pressure) {

// Equation taken from BMP180 datasheet (page 16):

// http://www.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf

// Note that using the equation from wikipedia can give bad results

// at high altitude. See this thread for more information:

// http://forums.adafruit.com/viewtopic.php?f=22&t=58064

return 44330.0 \* (1.0 - pow(pressure / MEAN\_SEA\_LEVEL\_PRESSURE, 0.190294957));

}

int RCtime(RCpin){

//wiringPiSetup();

pinMode(RCpin, INPUT);

int reading = 0;

delay(1);

int sensorState = digitalRead(RCpin);

while(1){

if (sensorState == 0){

reading += 1;

}

if (reading >= 1000) {

return 0;

}

if (sensorState != 0){

return 1;

}

}

return 0;

}

void buzz\_on(Rpin){

pinMode(Rpin, INPUT);

pinMode(Rpin, HIGH);

}

void buzz\_off(Rpin){

pinMode(Rpin, INPUT);

pinMode(Rpin, LOW);

}

int moistureSensor(){

if (RCtime(RCpin) == dry)

{

//printf("Sensor is still dry... \n");

buzz\_on(Rpin);

return 0;

}

if (RCtime(RCpin) == wet)

{

buzz\_off(Rpin);

//printf("Sensor is wet... \n");

return 1;

}

return 0;

}

void \*myThreadFun(void \*vargp)

{

while(1)

{

sleep(5);

//`if(datacounter%1000)

//{

printf("\nSending DATA\n");

result = sendData(email, temp, moisture, noise);

if (result == 0)

printf("Data has been sent!\n");

//return 0;

//}

}

}

int main (int argc, char \*argv[])

{

char hvacStatus[128] = "";

int padPin = 4;

int alreadyPressed = 0;

int value1, value2;

int data1, data2;

int counter = 0;

int step = 1; //1

int i = 0;

int offset = 85; //85

int deviation = 10; //10

int noiseCounter = 0;

int noiseDuration = 3; //20

int noiseOffCounter = 0;

int noiseOffDuration = 1000; //1000

int padPressed;

int datacounter;

if(argc < 3){

fprintf(stderr, "Usage: %s <hostname> <port>\n", argv[0]);

exit(1);

}

if (wiringPiSetup () == -1) {

printf("Error at wiringPiSetup()");

return 1 ;

}

pinMode(padPin, INPUT);

pinMode (0, OUTPUT) ; // aka BCM\_GPIO pin 17

pinMode (1, OUTPUT) ; // aka BCM\_GPIO pin 18

// Setup pcf8591 on base pin 120, and address 0x48

// printf("%d\n", ++step);

if (pcf8591Setup (PCF, 0x48) == -1) {

printf("Error at pcf8591Setup()");

return 1 ;

}

printf("Lowest noise level \t\t\t%d\n", offset);

printf("Data has been generalized with weight value of %d\n", step);

fd2 = socket\_connect(argv[1], atoi(argv[2]));

printf("\n Debug: %d\n ", fd2);

bme280\_init();

pthread\_t tid;

pthread\_create(&tid, NULL, myThreadFun, NULL);

//pthread\_join(tid, NULL);

while(1)

{

//datacounter++;

stream\_sensor\_data();

moistureread = moistureSensor();

padPressed = digitalRead(padPin);

if(padPressed)

{

if(alreadyPressed){}

else

{

strcpy(hvacStatus, "Pressed");

printf("\nSHUTDOWN OF PROGRAM INITIATED");

//alreadyPressed = 1;

return 0;

}

}

else

{

alreadyPressed = 0;

strcpy(hvacStatus, "Unpressed");

}

data1 += analogRead (PCF + 0); //white - local

data2 += analogRead (PCF + 1); //yellow - remote

counter++;

if(noise) {

noiseOffCounter++;

}

temp = temperature;

moisture = moistureread;

if(counter == step ) {

counter = 0;

value1 = round((double)data1 / step);

value2 = round((double)data2 / step);

data1 = 0;

data2 = 0;

if ((value1 < offset) || (value2 < offset)) {

if( abs (value1 - value2) < deviation) {

if (++noiseCounter > noiseDuration) {

noiseCounter = 0;

noiseOffCounter = 0;

noise = 1;

}

}

else {

if(noise) {

if(noiseOffCounter > noiseOffDuration) {

noiseOffCounter = 0;

noise = 0;

}

}

else if( --noiseCounter < 0) {

noiseCounter = 0;

}

}

if(noise) {

printf("\n\t%s\tNOISE !!!! #1:%3d #2:%3d Temperature: %4.2f Moisture: %d", hvacStatus, value1, value2, temperature, moistureread);

//both microphones are sensing noise - red

digitalWrite (0, LOW);

digitalWrite (1, HIGH);

}

else {

printf("\n\t%s\tVoice In!! #1:%3d #2:%3d Temperature: %4.2f Moisture: %d", hvacStatus, value1, value2, temperature, moistureread);

// Voice in at any of microphones - green

digitalWrite (0, HIGH);

digitalWrite (1, LOW);

}

for(i = offset; (i > offset - value1) && (i > 0); i--) {

printf(" ");

}

for(i = offset; i > value1; i--) {

printf("-");

}

printf("|");

for(i = offset; i > value2; i--) {

printf("+");

}

}

else {

if(!noise) {

digitalWrite (0, LOW);

digitalWrite (1, LOW);

}

}

}

}

shutdown(fd2, SHUT\_RDWR);

close(fd2);

return 0;

}