Room Management Web Application and Movement and Temperature Sensors

A Senior Project Report

Presented to

the Faculty of California Polytechnic State University
San Luis Obispo

In Partial Fulfillment of the Requirements for

the Degree Bachelor of Science in Computer Engineering

By

Visalbotr Chan
Huy Duong

March 2020
Abstract

There are three main parts of this system: micro-controller, database, and website. Micro-controller detects motion of people walking in and out and It also measures room temperature and humidity in a confined space then updates collected data to the database. Our system’s database contains 6 main columns: room number, room capacity, number of students, temperature in Celsius, humidity in percent and date created. Finally, this database is queried by the website to display the information on the webpage. Users could also navigate on our site to check the most and least occupy rooms, and they can also search for a specific room. Users can use this information to utilize rooms accordingly.
Table of Contents

I. Introduction 2
   a. Client 3
   b. Stakeholders 3
   c. Framed insight and opportunities 3
   d. Project Goals and Objective 3
   e. Project Deliverable 3
   f. Project Outcome 3

II. Background 4

III. Formal Project Definition 4

IV. Design 5
   a. Sensors 6
   b. Database 9
   d. Product 10

V. System Testing and Analysis 14

VII. Reflection 14

VIII. Appendices 14
   a. Bill of Materials: 14
   b. Datasheet 15
   c. Code List 15
   d. Test Cases 25
I. Introduction

Over summer 2019, I started to brainstorm ideas for my senior project and I wanted this project to showcase my knowledge in both hardware and software, as well as, an application that is useful for the Cal Poly community. I have been at Cal Poly for 3 years and finding a place to study is not the easiest thing especially during midterms and finals weeks. Therefore, I want to build a system to help students save time and utilize the study spaces at Cal Poly. This system will allow students to check how busy a room is, as well as, the temperature and humidity of it.

a. Client

We do not have a client or customer for this project. However, I could see that in the future this project could be used to implement in another area outside classroom scope.

b. Stakeholders

Our main stakeholders for this project is all Cal Poly students and other people who would want to utilize rooms at Cal Poly.

c. Framed insight and opportunities

We have talked to several students on campus about this project and they would love to use it to save their time.

d. Project Goals and Objective

Our goal is to build a prototype system that can showcase the users how the system can be implemented. We would like to be able to use sensors to detect the direction of the students and that sensor can communicate to a database which then will be used to be displayed on a website.

e. Project Deliverable

There are three main deliverable parts of this project, sensors implementation, database, and a web front-end.

f. Project Outcome

We were able to implement sensors that detect students movement and a database that bridges between the sensor and the web page. The web page displays all
the rooms with total students counts, temperature, and humidity. It also allows users to search for specific rooms, as well as, list the list or most crowded rooms.

II. Background

This project is inspired by a senior project, Waitz, that was done at the University of California San Diego in which they detect Bluetooth and Wi-Fi signals through raspberry pi in the area to determine the occupancy of an area. As right now, Waitz has been implemented in four universities and it is available on Android and iOS stores.

Despite the great success of Waitz, we want to tackle two main concerns with Waitz. First, the service requires students' device information through wireless communication. Second, the full feature service is only available on mobile platforms. Therefore, we have come up with sensors integration and a web application to solve the two concerns above. We are using sensors to detect the motion of the students; thus, we eliminate gathering any personal information from the users since it treats each person equally. On top of that, our application is widely adaptable on both mobile and web platforms.

III. Formal Project Definition

This project builds specifically for Cal Poly students and there are 3 main features that we would like to fulfill. Below are the customer requirements and engineering requirement for this project:

- Customer Requirements:
  - Check real-time occupancy, temperature, and humidity of a room
  - Sort every room in terms of occupancy
  - Search for a specific room
- Engineering Requirements:
  - Our system gathers data from users by two sonar sensors on a door, and every time it detects motion it will send information to a corresponding room in the database. The website queries information from the database and displays it to users.

<table>
<thead>
<tr>
<th>Spec. number</th>
<th>Parameter Description</th>
<th>Requirement or Target with units</th>
<th>Tolerance</th>
<th>Risk</th>
<th>Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Power</td>
<td>5 W</td>
<td>MAX</td>
<td>L</td>
<td>A, T</td>
</tr>
<tr>
<td>2</td>
<td>Code size</td>
<td>15 MB</td>
<td>MIN</td>
<td>L</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>Response Time Sonar Sensor</td>
<td>0.5 s</td>
<td>MAX</td>
<td>H</td>
<td>A, T</td>
</tr>
<tr>
<td>---</td>
<td>---------------------------</td>
<td>-------</td>
<td>-----</td>
<td>---</td>
<td>-----</td>
</tr>
<tr>
<td>4</td>
<td>Temperature and Humidity Sensor</td>
<td>5s</td>
<td>MIN</td>
<td>L</td>
<td>A,T</td>
</tr>
</tbody>
</table>

- End-user personas:
  - End-users are mainly Cal Poly students who look for a less crowded room to study or want to know the temperature or humidity of a specific room. End-users could access the webpage via https://radiant-sea-21897.herokuapp.com/ and will be able to browse through classes' stats.

**IV. Design**

There are 3 main parts of this project including sensors, database, and website. The sonar sensor is actively waiting for motions at the door. Once it registers the direction of a person walking in or walking out, it will update the information to the database. Then, the website will query all the information to show on the front page. The source code of this project could be found on a github link: https://github.com/vchan14/Room-Management-Web-Application-and-Movement-and-Temperature-Sensors.git
a. Sensors
i) Sonar Sensors

Two sonar sensors are used in each room to detect the direction of the students. The order of the triggers determine the direction of the students. In order to detect the presence of students, both sensors keep emitting waves and waiting for them to bounce back from a person. If there isn't anyone present, the wave will travel for 200cm from the top of the door frame to the floor. If there is, the wave will travel less than 50cm assuming our control student has height higher than 150cm (figure 2).
In figure 3, shows the connection between a sonar sensor with a raspberry pi. A voltage divider is needed because the output of the sonar Echo pin is 5V and it needs to be stepped down to 3.3V. When a motion is detected, raspberry pi will send a request to the database to update the information for the specific room.

**ii) Humidity and Temperature Sensor**

DHT11 sensor is used to get information about temperature and humidity. The information gets updated every 5 seconds and will be updated to the database as it gets new data. Below, figure 4, is the schematic between raspberry pi and the DHT11 sensor.
iii) LCD Live Data
Currently the LCD shows the number of students present in the enclosed space.
b. Database

Every data is stored in a table called "Rooms" which contains seven different columns includes:

1. id - primary key
2. room_number - String. ex: "14-237"
3. room_capacity - Integer. ex: 40
4. number_of_students - Integer. ex: 10
5. temperature_in_celsius - Integer. ex: 24
6. humidity_in_percent - Integer. ex: 40
7. date_created - Date. ex: 2020-01-27 19:10:53.69973

Postgresql is used to manage this database and it is hosted on Heroku platform.

<table>
<thead>
<tr>
<th>Modify</th>
<th>id</th>
<th>room_number</th>
<th>room_capacity</th>
<th>number_of_students</th>
<th>temperature_in_celsius</th>
<th>humidity_in_percent</th>
<th>date_created</th>
</tr>
</thead>
<tbody>
<tr>
<td>edit</td>
<td>1</td>
<td>14-237</td>
<td>35</td>
<td>1</td>
<td>24</td>
<td>29</td>
<td>2020-01-27 19:10:53.69973</td>
</tr>
<tr>
<td>edit</td>
<td>2</td>
<td>14-300</td>
<td>50</td>
<td>4</td>
<td>22</td>
<td>33</td>
<td>NULL</td>
</tr>
<tr>
<td>edit</td>
<td>25</td>
<td>20-001</td>
<td>100</td>
<td>50</td>
<td>20</td>
<td>20</td>
<td>2020-02-20 10:00:43.570472</td>
</tr>
<tr>
<td>edit</td>
<td>36</td>
<td>20-002</td>
<td>100</td>
<td>40</td>
<td>23</td>
<td>10</td>
<td>2020-02-20 10:00:43.855312</td>
</tr>
<tr>
<td>edit</td>
<td>37</td>
<td>20-003</td>
<td>100</td>
<td>0</td>
<td>23</td>
<td>40</td>
<td>2020-02-20 10:00:44.135601</td>
</tr>
<tr>
<td>edit</td>
<td>38</td>
<td>20-001</td>
<td>100</td>
<td>50</td>
<td>20</td>
<td>20</td>
<td>2020-02-20 10:01:01.91267</td>
</tr>
<tr>
<td>edit</td>
<td>39</td>
<td>20-002</td>
<td>100</td>
<td>40</td>
<td>23</td>
<td>10</td>
<td>2020-02-20 10:01:02.191478</td>
</tr>
<tr>
<td>edit</td>
<td>40</td>
<td>20-003</td>
<td>100</td>
<td>0</td>
<td>23</td>
<td>40</td>
<td>2020-02-20 10:01:02.475804</td>
</tr>
<tr>
<td>edit</td>
<td>41</td>
<td>20-001</td>
<td>100</td>
<td>50</td>
<td>20</td>
<td>20</td>
<td>2020-02-20 10:01:01.423985</td>
</tr>
<tr>
<td>edit</td>
<td>42</td>
<td>20-002</td>
<td>100</td>
<td>40</td>
<td>23</td>
<td>10</td>
<td>2020-02-20 10:01:01.721802</td>
</tr>
<tr>
<td>edit</td>
<td>43</td>
<td>20-003</td>
<td>100</td>
<td>0</td>
<td>23</td>
<td>40</td>
<td>2020-02-20 10:01:01.999195</td>
</tr>
<tr>
<td>edit</td>
<td>44</td>
<td>20-004</td>
<td>100</td>
<td>10</td>
<td>23</td>
<td>20</td>
<td>2020-02-20 10:01:11.278176</td>
</tr>
<tr>
<td>edit</td>
<td>45</td>
<td>20-005</td>
<td>100</td>
<td>20</td>
<td>23</td>
<td>10</td>
<td>2020-02-20 10:01:11.55611</td>
</tr>
<tr>
<td>edit</td>
<td>46</td>
<td>20-006</td>
<td>100</td>
<td>13</td>
<td>23</td>
<td>30</td>
<td>2020-02-20 10:01:11.830333</td>
</tr>
<tr>
<td>edit</td>
<td>47</td>
<td>20-007</td>
<td>100</td>
<td>0</td>
<td>23</td>
<td>30</td>
<td>2020-02-20 10:01:12.112784</td>
</tr>
<tr>
<td>edit</td>
<td>48</td>
<td>20-008</td>
<td>100</td>
<td>30</td>
<td>23</td>
<td>40</td>
<td>2020-02-20 10:01:12.391247</td>
</tr>
<tr>
<td>edit</td>
<td>49</td>
<td>14-001</td>
<td>40</td>
<td>10</td>
<td>25</td>
<td>30</td>
<td>2020-02-20 10:07:46.726602</td>
</tr>
<tr>
<td>edit</td>
<td>50</td>
<td>50-001</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>2020-02-20 10:07:46.002285</td>
</tr>
</tbody>
</table>

Figure 6: Rooms Table


The front page of the website shows all the classrooms that are in the database. There are five main columns that show room number, number of students, temperature, humidity, and crowd level. Users can also utilize the search bar to search for a specific class by typing [building number]-[room-number]. There are 4 other tabs on top of the search bar such as show all, top 5 rooms, Worst 5 Rooms, and Map. The top and worst 5 rooms show the first
most and the least occupancy rooms in the database. The map tab will allow users to see their current location on campus.

The front page is built from Flask, HTML, CSS, and JavaScript and it is hosted on heroku platform and can be accessed through the link above. Below, figure 6, is the screenshot of the front page website.

![Figure 6: Front page of the website](image)

<table>
<thead>
<tr>
<th>Room</th>
<th>Students</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Crowd Bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-237</td>
<td>1</td>
<td>24</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>14-300</td>
<td>4</td>
<td>22</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>20-001</td>
<td>50</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>20-002</td>
<td>40</td>
<td>23</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20-003</td>
<td>0</td>
<td>23</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>20-001</td>
<td>50</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>20-002</td>
<td>40</td>
<td>23</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20-003</td>
<td>0</td>
<td>23</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>20-001</td>
<td>50</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>20-002</td>
<td>40</td>
<td>23</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20-003</td>
<td>0</td>
<td>23</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>20-004</td>
<td>10</td>
<td>23</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>20-005</td>
<td>20</td>
<td>23</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20-006</td>
<td>13</td>
<td>23</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>20-007</td>
<td>0</td>
<td>23</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>20-008</td>
<td>30</td>
<td>23</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7: Front page of the website**

d. **Product**

Our group uses TinkerCAD to draw our 3D design closure box. Figures 8-10 show the different view of the box. Note: The orange lid is at the top of the box and two sonar sensors were
placed at the bottom of the box. Temperature and humidity sensor has been tested with raspberry pi; however, it hasn’t been integrated into the closure of the system.

Figure 8: Side view of the system
Figure 9: Bottom view of the system box
Figure 10: Inside view of the enclosure box
V. System Testing and Analysis

Regarding website testing, two different methods have been used to determine the reliability of the system. Python script was used to replicate multiple post requests from raspberry pis to the database. Selenium was used to test the functionality of the front page by replicating the user navigation from one tab to another.

VI. Conclusion and Future Work

Throughout these two quarters, the prototype system could detect students’ movement and acquire temperature and humidity. Both database and front-end page are currently live on the Heroku platform. There are two main APIs that allow raspberry pi to communicate to the database and web page to query information from the database.

There are several improvements that could be done for future work. A second motion sensor such as thermal camera or lidar could be used concurrently with the sonar sensors to increase the accuracy of the system. On top of that, a distance calculation could be implemented on the website to allow users to know how far it takes to go from their current location to a specific room. In addition, the temperature and humidity sensor should be added into the enclosure box. The LCD can show the temperature and humidity live data for students inside the room.

VII. Reflection

These last two quarters have allowed us to explore what we want to learn and do in our career. We had run into several issues both designs and choosing tools. We spent about 3 weeks trying to interface a flight sensor and then found that it is not compatible with raspberry pi. Moreover, we had to analyse our sonar sensor algorithm several times to optimize the accuracy of the detection.

VIII. Appendices

a. Bill of Materials:

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Supplier</th>
<th>Quantity</th>
<th>Unit Price</th>
<th>Extended Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Raspberry Pi 3 Model B Complete Starter Kit</td>
<td>CanaKit</td>
<td>1</td>
<td>74.99 $</td>
<td>74.99</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Brand</td>
<td>Quantity</td>
<td>Cost</td>
<td>Total</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------------------------------</td>
<td>------------</td>
<td>----------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>2</td>
<td>Ultrasonic Module (HC-SR4)</td>
<td>Elegoo</td>
<td>2</td>
<td>2.2 $</td>
<td>4.4$</td>
</tr>
<tr>
<td>3</td>
<td>Temperature and Humidity Module (DHT11)</td>
<td>HiLetgo</td>
<td>1</td>
<td>2.6$</td>
<td>2.6$</td>
</tr>
<tr>
<td>4</td>
<td>Breadboard</td>
<td>Elegoo</td>
<td>1</td>
<td>3.0$</td>
<td>3.0$</td>
</tr>
<tr>
<td>5</td>
<td>Male to Male, Female to Female, Male to Female 120 Jumper Wires</td>
<td>Elegoo</td>
<td>1</td>
<td>6.98 $</td>
<td>6.98$</td>
</tr>
<tr>
<td>6</td>
<td>Resistors</td>
<td>Elegoo</td>
<td>2</td>
<td>0.10 $</td>
<td>0.20$</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>92.17 $</strong></td>
</tr>
</tbody>
</table>

b. Datasheet

**DHT11 - Temperature sensor:**

**HC-SR04 - Sonar Sensor:**
https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

**Raspberry Pi 3 Model B:**

c. Code List


i. **movement_detect.py**

sonar sensor api, allow the sensors to communicate with the database.

```python
import RPi.GPIO as GPIO
import time
import sys
```
sys.path.append('/home/pi/Desktop/senior-project-who-are-there/deploy_app')
GPIO.setmode(GPIO.BCM)
from database import UpdateDB

class ULTRASONIC_SENSOR():
    def __init__(self, trig_gpio, echo_gpio):
        self.trig = trig_gpio
        self.echo = echo_gpio
        # set trig for output
        # set echo for input
        GPIO.setup(trig_gpio, GPIO.OUT)
        GPIO.setup(echo_gpio, GPIO.IN)

    def calculateDistance(self, pulse_end, pulse_start):
        # we know that sound's speed is 34,000cm/s
        # distance = time_taken * 34,000 / 2
        # divide by two because the sound needs to travel back and forth
        return ((pulse_end - pulse_start) * 17150)

    def detect(self):
        current_dist = self.distance()
        if(current_dist < 50):
            return True
        else:
            return False

    def distance(self):
        time.sleep(0.01)
        GPIO.output(self.trig, True)
        time.sleep(0.00001)
        GPIO.output(self.trig, False)
        pulse_start = 0
        pulse_end = 0

        while GPIO.input(self.echo) == 0:
            pulse_start = time.time()

        while GPIO.input(self.echo) == 1:
            pulse_end = time.time()

        return self.calculateDistance(pulse_end, pulse_start)

# assign GPIO
TRIG_A = 23
ECHO_A = 24
TRIG_B = 22
ECHO_B = 27

sensor_A = ULTRASONIC_SENSOR(TRIG_A, ECHO_A)
sensor_B = ULTRASONIC_SENSOR(TRIG_B, ECHO_B)

try:
    time_diff = 0
    time_A = 0
    time_B = 0
    # create an obj class to modify the database
    obj = UpdateDB()

    while(True):
        if(sensor_A.detect() == True):
            print("A is detected")
            time_A = time.time()
            while(1):
                if(sensor_B.detect() == True):
                    print("B is detected")
                    print("Going in")
                    obj.incrementNumberStudent("14-237")
                    time.sleep(0.8)
                    break
                else:
                    time_diff = time.time() - time_A
                    if(time_diff > 5):
                        break
            if(sensor_B.detect() == True):
                print("B is detected")
                time_B = time.time()
                while(1):
                    if(sensor_A.detect() == True):
                        print("A is detected")
                        print("Going out")
                        obj.decrementNumberStudent("14-237")
                        time.sleep(0.8)
                        break
                    else:
                        time_diff = time.time() - time_B
                        if(time_diff > 5):
                            break

# If there is a Keyboard Interrupt (when you press ctrl+c), exit the program and cleanup
except KeyboardInterrupt:
    print("Cleaning up!")
import RPi.GPIO as GPIO
import dht11
import time
import datetime
import sys
sys.path.append('/home/pi/Desktop/senior-project-who-are-there/deploy_app')
from database import UpdateDB

# initialize GPIO
GPIO.setwarnings(True)
GPIO.setmode(GPIO.BCM)

# read data using pin 14
instance = dht11.DHT11(pin=14)

def getTempHum():
    while True:
        result = instance.read()
        if result.is_valid():
            time.sleep(2)
            return(int(result.temperature), int(result.humidity))

def main():
    # create an obj class to modify the database
    obj = UpdateDB()
    while True:
        myTempHum = getTempHum()
        print("Temperature is {}").format(myTempHum[0]))
        print("Humidity is {}").format(myTempHum[1]))
        obj.updateTemperature("14-237", myTempHum[0])
        obj.updateHum("14-237", myTempHum[1])

if __name__ == '__main__':
    main()
### iii. database.py

It is responsible for populating the table and contains all the APIs that are used by sonar and temperature and humidity sensors.

```python
from datetime import datetime
from flask import Flask, render_template, request
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SQLALCHEMY_DATABASE_URI'] = #database_uri

db = SQLAlchemy(app)

# create a table

class Rooms(db.Model):
    __tablename__ = "Rooms"
    id = db.Column(db.Integer, primary_key=True)
    room_number = db.Column(db.String(50), nullable=False)
    room_capacity = db.Column(db.Integer, nullable=False)
    number_of_students = db.Column(db.Integer)
    temperature_in_celsius = db.Column(db.Integer)
    humidity_in_percent = db.Column(db.Integer)
    date_created = db.Column(db.DateTime, default=datetime.now)

class UpdateDB:

    def createRoom(self, room_number, room_capacity, number_of_students, temperature_in_celsius, humidity_in_percent):
        room = Rooms(room_number=room_number,
                      room_capacity=room_capacity,
                      number_of_students = number_of_students,
                      temperature_in_celsius = temperature_in_celsius,
                      humidity_in_percent = humidity_in_percent)

        db.session.add(room)
        db.session.commit()
```
return False

# function that increment number students in a particular room
def incrementNumberStudent(self, room_number):
    room = Rooms.query.filter_by(room_number = room_number).first()
    room.number_of_students +=1
    db.session.commit()
    return False

# function that decrement number students in particular room
def decrementNumberStudent(self, room_number):
    room = Rooms.query.filter_by(room_number = room_number).first()
    room.number_of_students -=1
    db.session.commit()
    return False

# function that query the number students in a particular room
def numberStudent(self, room_number):
    numberStudent = Rooms.query.filter_by(room_number = room_number).first().number_of_students
    return numberStudent

def updateTemperature(self, room_number, temp_c):
    room = Rooms.query.filter_by(room_number=room_number).first()
    room.temperature_in_celsius = temp_c
    db.session.commit()
    return False

def updateHum(self, room_number, hum):
    room = Rooms.query.filter_by(room_number=room_number).first()
    room.humidity_in_percent = hum
    db.session.commit()
    return False

# return a dictionary that have key(room number) and value(number of students in the room)
def classRooms(self):
    rooms = Rooms.query.all()
    return rooms

def main():
    db.create_all() # create all table
# create an object to create room rows
obj = UpdateDB()

obj.createRoom("14-001", 40, 10, 25, 30)
obj.createRoom("50-001", 30, 10, 10, 20)

if __name__ == '__main__':
    main()

---

**iv. app.py**
This is a main web application that is responsible for all the http requests.

```python
from datetime import datetime
from flask import Flask, render_template, request
from flask_sqlalchemy import SQLAlchemy
from database import *
from flask_bootstrap import Bootstrap
import re

app = Flask(__name__)
Bootstrap(app)

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SQLALCHEMY_DATABASE_URI'] = #database_uri
db = SQLAlchemy(app)

@app.route("/")
def index():
    obj = UpdateDB()
    temp = obj.classRooms()
    return render_template("index.html", rooms=temp)

@app.route("/search")
def search():
    search_term = "^" + request.args.get('query')
    rooms = Rooms.query.all()
    output = []
    for room in rooms:
        if(re.search(search_term, room.room_number)):
output.append(room)

return render_template("index.html", rooms=output)

@app.route("/map")
def map():
    return render_template("map.html")

@app.route("/top5")
def topfive():
    rooms = Rooms.query.all()
    rooms_crowd = []
    output = []
    for room in rooms:
        rooms_crowd.append((room, room.number_of_students / room.room_capacity))
    rooms_crowd.sort(key=lambda x:x[1])
    for i in range(5):
        output.append(rooms_crowd[i][0])
    return render_template("index.html", rooms = output)

@app.route("/worst5")
def worstfive():
    rooms = Rooms.query.all()
    rooms_crowd = []
    output = []
    for room in rooms:
        rooms_crowd.append((room, room.number_of_students / room.room_capacity))
    rooms_crowd.sort(key=lambda x:x[1])
    for i in range(5):
        output.append(rooms_crowd[len(rooms_crowd)-1-i][0])
    return render_template("index.html", rooms = output)

@app.route("/trends")
def trends():
    return render_template("trends.html")
@app.route("/monday")
def monday():
    return render_template("monday.html")
if __name__ == '__main__':
    app.run(debug=True)

v. index.html
This is the front page of the website.

{% extends "navBar.html" %}

{% block content %}

<div class="container">
    <h3 class="page-header" id="Title" > Cal Poly San Luis Obispo Classes</h3>

    <button type="button" class="btn btn-default">
        <a href={{ url_for('index') }}>
            Show All
        </a>
    </button>

    <button type="button" class="btn btn-default">
        <a href={{ url_for('topFive') }}>
            Top 5 Rooms
        </a>
    </button>

    <button type="button" class="btn btn-default">
        <a>

    </button>

{% endblock %}
Worst 5 Rooms

<href={{ url_for('worstFive') }}>Worst 5 Rooms</a>
</button>

<button type="button" class="btn btn-default">
  <a href={{ url_for('map') }}>Map</a>
</button>

<div class="col-lg">
  <form class="input-group" method="GET" action="search" >
    <input type="text" class="form-control" name="query" id="query" placeholder="Search for...">
    <span class="input-group-btn">
      <button class="btn btn-default" type="submit">Go!</button>
    </span>
  </form>
</div>

<!--We need to search for classroom-->

<table class="table table-bordered table-condensed table-striped">
  <thead>
    <tr>
      <th scope="col">Room</th>
      <th scope="col">Students</th>
      <th scope="col">Temperature</th>
      <th scope="col">Humidity</th>
      <th scope="col">Crowd Bar</th>
    </tr>
  </thead>
  <tbody>
    {% if room != None %}
    {% for room in rooms %}
    <tr>
      <td>Room</td>
      <td>Students</td>
      <td>Temperature</td>
      <td>Humidity</td>
      <td>Crowd Bar</td>
    </tr>
    {% endfor %}
    {% endif %}
  </tbody>
</table>
<tr>
<!-- Set variable in Jinja-->
{% set crowd_percent = (room.number_of_students *100 //room.room_capacity) %}
<td>{{room.room_number}}</td>
<td>{{room.number_of_students}}</td>
<td>{{room.temperature_in_celsius}}</td>
<td>{{room.humidity_in_percent}}</td>
<td><div class="progress">
    <div class="progress-bar" role="progressbar" aria-valuemin="0" aria-valuemax="100" style="width: {{crowd_percent}}%;"></div>
</div></td>
</tr>
{% endfor %}
{% endif %}
</tbody>
</table>
</div>

{% endblock %}


d. Test Cases

i. simulate the usage of the sensors APIs in real life

def main():

    # create a class to create room row
    obj = UpdateDB()

    obj.createRoom("14-001", 40, 10, 25, 30)

    # test increment and decrement students in room 14-001
    while (True):
        for i in range(50):
obj.updateTemperature("14-001", i)
obj.updateHum("14-001", i)
obj.incrementNumberStudent("14-001")
if (i % 2):
    obj.decrementNumberStudent("14-001")
sleep(1)

for i in range(10):
    obj.decrementNumberStudent("14-001")
sleep(1)

ii. front-end test

from selenium import webdriver
from time import sleep
from selenium.webdriver.common.keys import Keys
import unittest
class Bot():
    def __init__(self):
        self.driver = webdriver.Chrome()

    def giveElement(self, path):
        return self.driver.find_element_by_xpath(path)

    def go_to(self, webiste):
        self.driver.get(webiste)

    def showAll(self):
        show_all_button_path = '/html/body/div[2]/button[1]'
        show_all_button = self.giveElement(show_all_button_path)
        show_all_button.click()

    def topFive(self):
        top_five_button_path = '/html/body/div[2]/button[2]'
        top_five_button = self.giveElement(top_five_button_path)
        top_five_button.click()

    def worstFive(self):
        worst_five_button_path = '/html/body/div[2]/button[3]'
        worst_five_button = self.giveElement(worst_five_button_path)
worst_five_button.click()

def searchRoom(self, room):
    search_box_path = '//*[@id="query"]'
    search_box = self.giveElement(search_box_path)
    search_box.send_keys(room)

    search_button_path = '/html/body/div[2]/div/form/span/button'
    search_button = self.giveElement(search_button_path)
    search_button.click()

def tearDown(self):
    self.driver.close()

def main():
    bot = Bot()

    # go to the website
    bot.go_to('https://radiant-sea-21897.herokuapp.com/

    # Do the following tests 10 times
    for i in range(10):
        # check the top 5 Rooms
        sleep(2)
        bot.topFive()

        # check the worst 5 rooms
        sleep(2)
        bot.worstFive()

        # show all the room
        sleep(2)
        bot.showAll()

        # search for room 14-237
        sleep(2)
        bot.searchRoom("14-237")

        bot.tearDown()

if __name__ == "__main__":
    main()