Deciwatcher

Joseph Spillers

IUPUI

Abstract

Workspaces generally are designed to maximize productivity. Having a monitoring system could allow administrators of an environment a more thorough understanding of the peak times for productivity for their workers. A web+IoT based platform was developed to monitor this occurrence.

Keywords : NodeJS, Express, React, SQL, Decibles, Startup, Arduino, ESP8266

Table of Contents

1. Introduction

1.1 Background

Rationale. Workspaces generally are designed to maximize productivity. Sound level, in this case study from the context of intensity or Decibels, is the piece of interest for this study. According to an overarching study monitoring optimal sound environments, a conclusion was made that , "In an office setting, he recommends a large, open space be below 70 dB, and background noise from mechanicals and speaking to be around 45 to 55 dB to keep people happy and healthy" (Cravens 16).

Student Perspective. Of this ambient noise level, a coffee shop researcher noticed students specifically "uniformly favour the sonic environment of the café and feel detached and leery of rainforest ambiences"(DROUMEVA 124). Watching for the maximum acceptable noise level in a workplace environment, a study was done monitoring the health of nurses caring for young children. The study concluded "occupied facilities ranged from 60dBA to 89dBA … While the majority of participants did not report ringing in their ears, all employees (n=2) working in the facility with minimal acoustic considerations reported ringing in their ears." (Pope 11).

Starting Point. Crying children tend to not exist in a workplace environment, but it is useful to keep this figure in mind, as this could be used to represent the dipping point in being the least effective at maintaining productivity.

1.2 Objective

The goal is to create a device which will monitor the ambient volume of a given location, and work with a collection of other devices to build a sort of map which an end user could view the current volume of a location, as well as the relative volume over time to view the historical impact of the 'most productive' locations in an environment, and the least.

1.3 Significance

As referenced in the background, noise level monitoring is a significant measure of the productivity of a workplace. Having a monitoring system could allow administrators of an environment a more thorough understanding of the peak times for productivity for their workers. Additionally, this could allow those who work in various volume environments most effectively an opportunity to self sort to where they would most prefer to work.

1.4 Related works or available technology

There exist a number of machines on the market today which monitor specific sounds and other recording devices, like the oft referenced "James Bond Bug", however this specific monitoring system has not been attempted in this particular library style context before.

2. Method and Implementation

2.1 Methods (if you tried multiple methods, method1, method2, ...)

Interconnection. Essentially the Deciwatcher Device (an Internet of Things [IoT] device, in this implementation an ESP8266) sends Wifi packets over a network to the Express Backend, which interfaces with a SQL database to store each packet. The Express backend also handles requests from a Web Front End written in React, and a Mobile Front End, written in React Native which has support for IOS and Android. The Mobile Front End connects directly to the IoT Platform and connects it to the Wifi network of interest.

SQL Structure. Figures 2 and 3 represent the decided SQL structure for interaction as well as a bit of sample data to demonstrate how it is intended to function. The *IoTSensors* Table stores the relative MAC of each physical device, as well as related information such as the Sensor Name, Location, and a picture of it. The *DBReadings* table stores individual readings sent out by the IoT Sensor in decibels with timestamps.

Express Backend. The Express Backend handles three major functions. Firstly, it receives POST packets from the Arduino device, and interprets them to store them into the *DBReadings* table. Secondly, it receives registration requests from the React Native application to register new IoT devices to the platform. Thirdly, it receives data requests from the React and React Native applications to receive a historical record of data received by all Arduino devices.

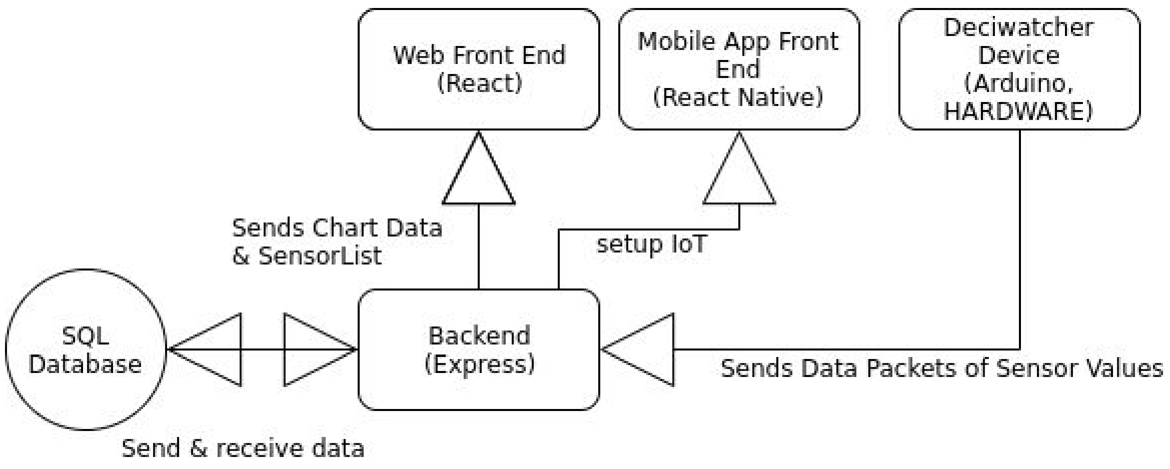
React. The React application needs to provide an intuitive web interface for a quick and easy viewing of all the Arduino sensors' histories. This needed to be fast, and clean. Figure 5 shows the current working implementation running a card interface, which means regardless of screen size it clips to the appropriate sizing. According to an article written by A. Raj (2018), utilization of the regression method to parallel the measured output vs expected output can achieve numbers very close to accuracy with incredibly cheap Arduino condenser microphones, so this approach was taken and is demonstrated in the code sample for the Arduino.

Arduino (ESP8266). This device needed to listen on a variable interval for the decibel level of the location it is situated. From that, every minute it calculates the average of the decibels recorded, and sends this to the Express backend. A diagram of this is located under Figure 4. The wifi network and interval need to be set manually on the Arduino programmer.

2.2 Computing environments

The Front End is a React node platform which must run on a server capable of this functionality. The backend is an Express node platform which must also run on a capable server. The IoT platform runs on the ESP8266 platform and runs standalone on power, utilizing the Arduino runtime and a wireless connection. Each device for simplicity sake runs on the same network and the same server (sans the IoT) however this is not necessary as each server could easily be run on something like Amazon's AWZ or Google's Firebase with just redirecting the fetch calls to the respective platform.

2.3 Diagram of processing


Web Front End (React)
Mobile App Front End (React Native)
Deciwatcher Device (Arduino, HARDWARE)
Sends Chart Data & SensorList
setup IoT
SQL Database
Backend (Express)
Sends Data Packets of Sensor Values
Send & receive data

2.4 Software implementation or program

The software application implementation can be found here : https://github.iu.edu/jdspille/capstone , With the main code for each section found under the respective folders.

Note the React Native Front end was abandoned as the interconnection of so many platforms left time a bit too short to tackle this portion of the project.

3. Implementation and experiments

3.1 Data acquisition or system construction

Data is gathered by the IoT sensor , processed by the referenced regression algorithm, and stored in the SQL database as mentioned in the implementation.

3.2 Data analysis, system implementation

The ambient volume of my room seems to stick around 50 dB, which is around what is to be expected. Representation of data becomes more difficult as the number of individual measurements increases. This was somewhat combated by reducing the number of measurements gathered per minute, however in the long term this will prove to need work to maintain efficiency.

3. 3 Results display in figure, image, demonstration, etc.

See Figures 2, 3, and 5 in Appendix.

3.4 Evaluation of system design, accuracy of analysis, processing time

Application implementation made use of OOP principles and didn't violate SOLID design constraints. All applications load under 5 seconds, and IoT device has no lag time, and minimizes individual runtime utilizing the system's 'delay' command. Decibel recordings are quite accurate as the mathematical mapping has a negligible deviation.

3.5 Problem found and discussion of future solution

Modification of the parameters of the runtime of the Arduino environment are only modifiable from the Arduino IDE, which is quite troublesome. It would be much better to be able to modify this from a mobile Android application. This would need to be programmed natively as it would need to connect over a soft AP to the Arduino and push Wifi connection details to the device.

4. Conclusion

Deciwatcher Device (an Internet of Things [IoT] device, in this implementation an ESP8266) sends Wifi packets over a network to the Express Backend, which interfaces with a SQL database to store each packet. The Express backend also handles requests from a Web Front End written in React, and a Mobile Front End, written in React Native which has support for IOS and Android. The Mobile Front End connects directly to the IoT Platform and connects it to the Wifi network of interest. Registering IoT devices requires manual input, but otherwise this is a complete solution.

References

Craven, V. D. (2018). The Role of Acoustics in Workplace Health: THE RIGHT EDUCATION AND MATERIALS CAN HELP IMPROVE WORKPLACE PRODUCTIVITY AND SATISFACTION. *Buildings*, (8), 16. Retrieved from https://search-ebscohost-com.proxy.ulib.uits.iu.edu/login.aspx?direct=true&db=edsgsr&AN=edsgcl.550997725&site=eds-live

DROUMEVA, M. (2017). SOUNDWORK: The Coffee-Office: Urban Soundscapes for Creative Productivity. *BC Studies*, (195), 119–127. Retrieved from https://search.ebscohost.com/login.aspx?direct=true&db=aph&AN=128514112&site=eds-live

Pope, Emily K.. (2018). Decibel Levels and Employee Perceptions of Noise Levels in Child Care Facilities. In *BSU Honors Program Theses and Projects.* Item 272. Retrieved from https://vc.bridgew.edu/honors_proj/272

Raj, A. (2018). Measure Sound/Noise Level in dB with Microphone and Arduino. Retrieved from https://circuitdigest.com/microcontroller-projects/arduino-sound-level-measurement

Appendix

Figures

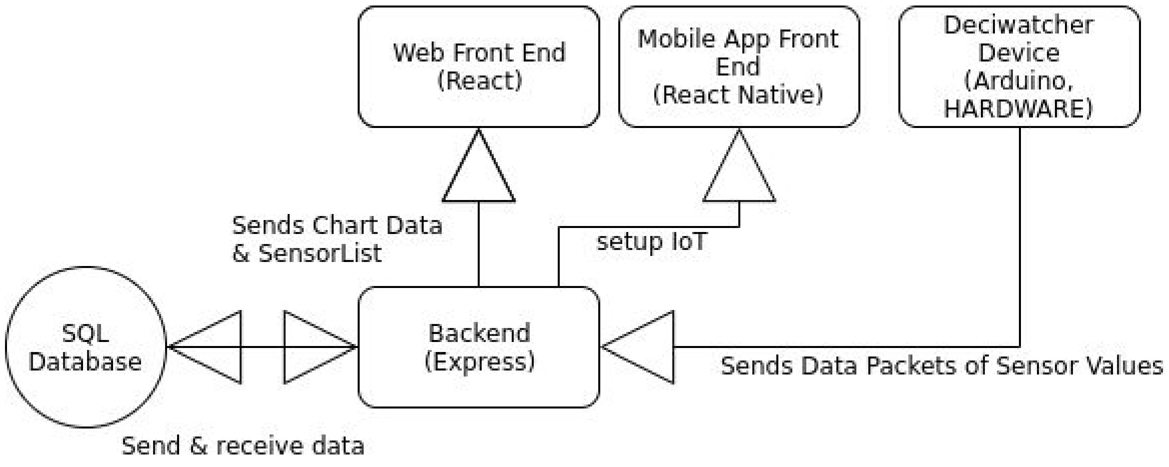
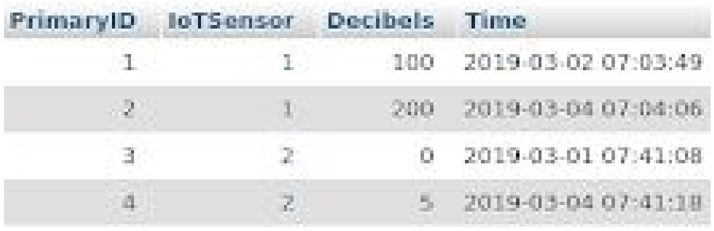
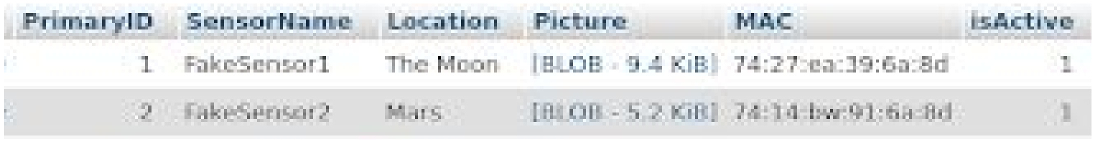


Figure 1 : Functional Implementation



PrimaryID	IoTSensor	Decibels	Time
1	1	100	2019-03-02 07:03:49
2	1	200	2019-03-04 07:04:06
3	2	0	2019-03-01 07:41:08
4	2	5	2019-03-04 07:41:18

Figure 2 : SQL Database, DBReadings



PrimaryID	SensorName	Location	Picture	MAC	isActive
1	FakeSensor1	The Moon	[BLOB - 9.4 KiB]	74:27:ea:39:6a:8d	1
2	FakeSensor2	Mars	[BLOB - 5.2 KiB]	74:14:bw:91:6a:8d	1

Figure 3 : SQLDatabase, IoTSensors

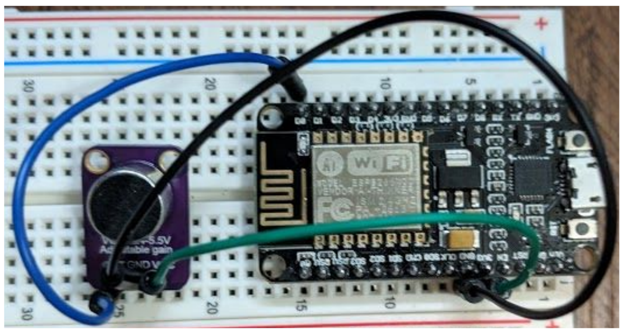
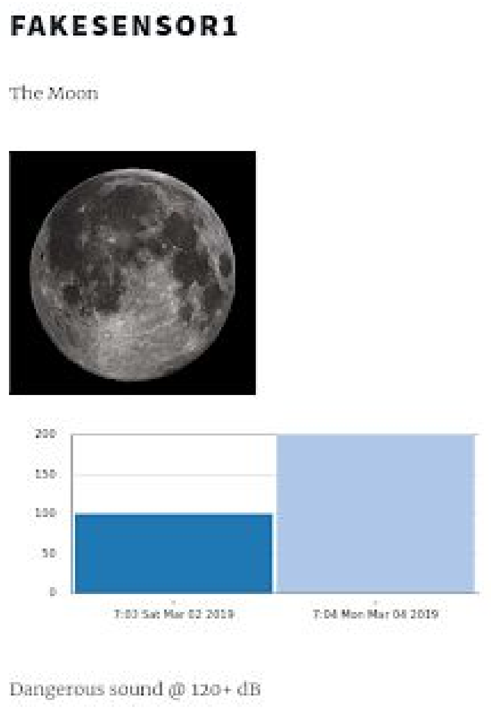


Figure 4 : Device Prototype 1


FAKESENSOR1

The Moon

Dangerous sound @ 120+ dB

Figure 5 : React Web Interface