IoT-SHM: Internet of Things: Structural Health Monitor

ECE4012 Senior Design Project

Section L3C, IoT SHM Team
Project Advisor, Dr. Vijay Madisetti

Julie Freeman, julieefreeman@gatech.edu, Team Leader
Reinaldo Cruz, reinaldo422@gatech.edu
Margaret Fitzharris, mfitzharris3@gatech.edu
Rasha El-Jaroudi, rhej3@gatech.edu
Douglas Sigelbaum, dougsig@gatech.edu

Submitted

April 29, 2015

Executive Summary

IoT-SHM is a structural health monitoring system based on the internet of things. The cost of the system varies depending on the number of nodes (sensors) placed on a structure and the number of structures being monitored, which would be determined by the team and the client. The base price for installation begins at $800 given a system with three sensors reading acceleration data and the yearly web services costs to the client would be $430. Due to the flexibility of the design, the number of cloud resources and the size of these resources would factor into this cost.

Sensors are placed on a structure in various locations to monitor readings of the client's choosing. Later in this report, types of readings are discussed that are often used to monitor the state of a structure. The prototype uses accelerometers. The sensor nodes are configured to communicate and send data to a gateway node. Each structure connected to IoT-SHM will have several sensors with just one gateway node. This node must be able to connect to the internet in order to send data to the analytics side of IoT-SHM.

This system relies on the assumption that the structure in question is deemed healthy at the time of installation. 'Healthy' data will be gathered and used to determine the state of the structure in real-time based on current sensor readings. Data is analyzed in the cloud and stored in cloud-hosted database. The system relays real-time data to the user in the form of a web application. The application displays the real-time readings from the sensors as well as whether or not the current state of the structure is deemed healthy or unhealthy as determined by analysis. The IoT-SHM prototype was tested using a clothes dryer as the controlled environment due to an easily identifiable and controllable healthy (off) state and unhealthy (on) state.

Table of Contents

Executive Summary ii

1 Introduction 1

1.1 Objective 1
1.2 Motivation 2
1.3 Background 2

2 Project Description and Goals 4

3 Technical Specifications & Verification 5

3.1 Quantitative Specifications 5
3.2 Qualitative Specifications 6

4 Design Approach and Details 6

4.1 Design Approach 6
4.2 Codes and Standards 12
4.3 Constraints, Alternatives, and Tradeoffs 13

5 Schedule, Tasks, and Milestones 17

6 Final Project Demonstration 18

6.1 Test Setup 18
6.2 Project Specifications Demonstrated 19

7 Marketing and Cost Analysis 21

7.1 Marketing Analysis 21
7.2 Cost Analysis 22

8 Conclusion 27

8.1 Current Status 27
8.2 Lessons Learned 27
8.3 Future Expansions 28

9 References.. 29

Appendix A……………………………………………………………………………… 32

Appendix B……………………………………………………………………………... 34

Structural Health Monitor: Internet of Things Implementation

1. Introduction

Team IoT-SHM requested $415 to develop an end-to-end prototype of a permanent Structural Health Monitoring system. The system will be compatible with various types of structures and various types of data analysis.

1.1 Objective

The objective of the IoT-SHM system was to use a network of sensors to identify deterioration or instability in a structure. The focus of this project was a network of sensors deployed on the structure followed by the data collection and analytics. A machine learning algorithm was used to analyze the collected data to determine the current status of the structure. The front-end data visualization displayed the current status/health of the structure and readings from the sensors.

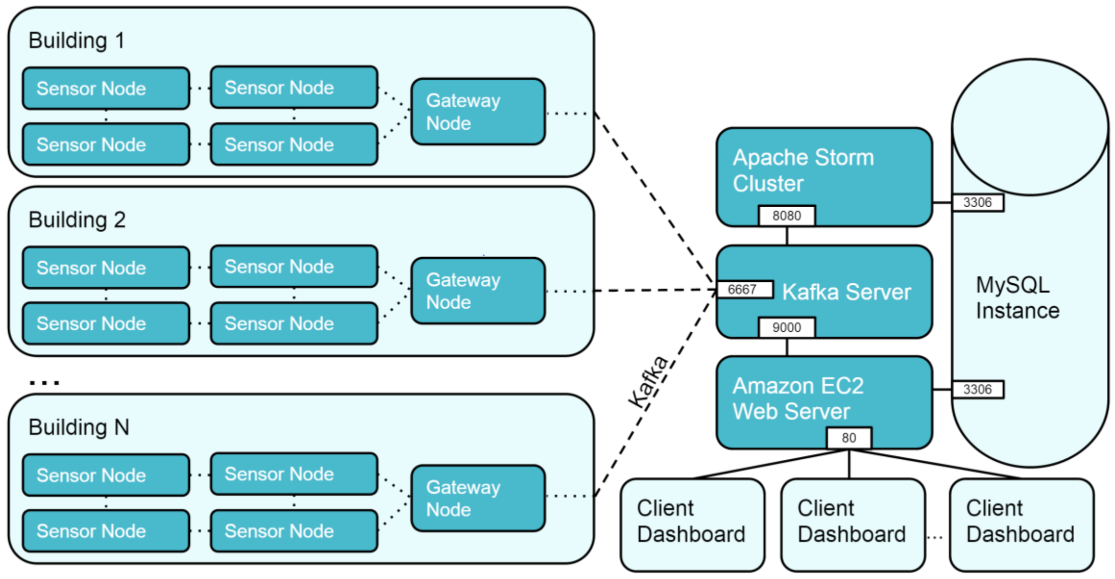


Figure 1. SHM system overview.

1.2 Motivation

IoT-SHM is a low cost, low-energy, end-to-end permanent SHM solution to determine the current and past health of a structure. Currently available SHM systems are expensive, with skeleton systems starting well above the overall cost of the complete system. A fully packaged system such as the Digitexx system costs approximately $10,000 [1]. This Digitexx permanent system records the system remotely and notifies the client when an event is triggered [1]. IoT-SHM is a cloud-based design that allowed its users more flexibility to fully monitor the structure's current state. It is adaptable to different types of data for monitoring the structure and can be customized per the customer's needs. Flexibility in terms of data also allows for IoT-SHM to be adaptable to structures of all kinds.

1.3 Background

1.3.1 Current SHM systems

Currently, commercial applications of structural health monitoring demand both permanent and portable monitoring. Permanent systems are for continuous long-term monitoring such as for months or years, while portable systems are for assessing the current conditions of the structure. Permanent systems allow for continuous monitoring and are commonly used for health monitoring, long term strain studies, vibration studies and fatigue monitoring. The current permanent structure health monitoring systems use accelerometers, temperature sensors, wind sensors, and pressure sensors [1].

1.3.2 Sensor Types and Accelerometers

There are three types of accelerometers which were considered in this design: piezoelectric, capacitive, and piezoresistive.

Piezoelectric:

According to the piezoelectric effect, when a force is applied, opposite charges accumulate on either side of the crystal. This voltage creates a capacitive effect, which can then be measured by piezoelectric accelerometers to determine the acceleration. Piezoelectric accelerometers are typically very precise, high cost, and used in very demanding scenarios [2].

Capacitive:

Capacitive accelerometers measure the acceleration through the change in capacitance when a force causes a displacement in the structure. These accelerometers are best applied in low cost and power situations and have a wide range of applications [2].

Piezoresistive:

When a force is applied to a piezoresistive material, the resistance varies accordingly according to the piezoresistive effect which can then be used to measure acceleration in piezoresistive accelerometers. These devices can measure acceleration as low as 0 and are often employed for high shock scenarios. They are best used in low frequency applications as their high frequency performance is limited [2].

1.3.3 Cloud Computing and Internet of Things

Cloud computing allows for applications and data to be easily accessible and dynamically managed. This allows systems to access resources without relying on specific machines, and for these systems to be updated efficiently [3]. Cloud computing also makes an Internet of Things (IoT) approach for connecting hardware devices to the internet cost effective and reasonable within the scope of this project [4]. This approach was appropriate for this project as the user would be able to monitor the structure without being restricted to a specific location in order to access the system. Users benefit from the pay-as-you-go approach which requires clients to pay for only as many resources as are used. This project relied on the cloud to maintain the data collected from the structural sensors and manipulate it using customized algorithms. In addition, the goal was to convert data collected from physical devices into an easily accessible and readable interface.

2. Project Description and Goals

The overall goal of the project was a prototype structural health monitor system that can be seamlessly extended to any structure. The minimum-viable-product was applying this system design to a structure using just one type of sensor. The entire system has three main components:

- Wireless mesh network of sensors that procured vibration information from the structure
- Cloud system that computed and stored the health of the structure
- Browser GUI that displayed real-time analytics from the structure

Technical goals:

- Sensors were self-sufficient in terms of power
    - Low power communication protocols and devices

  **Actual: Arduinos used for sensor nodes which are not the most power efficient devices available however were chosen for ease of development purposes.*
- Used a machine learning algorithm to create an accurate structural health analysis
- Kept hardware costs less than $500
- System did not throw false-positives after months of training

  **Actual: System was trained for 4 consecutive hours. This short time period was acceptable due to the controlled test environment.*

3. Technical Specifications

3.1 Quantitative Specifications

TABLE 1
Sensor Gateway Computer Specifications

Feature	Proposed Specifications	Demonstrated Specifications
Sensor node connection bandwidth	250 Kbps	250 Kbps
Messaging protocol	MQTT	Kafka (binary protocol over TCP)
RAM	512 Mb	512 Mb
Operating System	Linux-based	Linux-based
Power supply	5 V	5 V
Dimensions	86 mm x 60 mm	86 mm × 60 mm

TABLE 2
Sensor Node Specifications

Feature	Proposed Specifications	Demonstrated Specifications
Microcontroller clock speed	16 MHz	16 MHz
Flash memory	512 Kb	512 Kb
RAM	66 Kb	66 Kb
Sensor output data rate	200 kHz	200 kHz
Power supply	3.6 V - 15 V	3.6 V - 15 V
Dimensions	62 mm x 45 mm	62 mm x 45 mm
Mesh Network Protocol	802.15.4	802.15.4
Mesh Network Throughput	80 kbps	80 kbps

TABLE 3
Cloud Service Specifications

Feature	Proposed Specifications	Demonstrated Specifications
Size of Apache Storm cluster	20 nodes	1 node
Hours/day running instances	840 Hours (5 weeks)	24 hr/day (2 instances)
Maximum sensor data stored	2 TB	2 TB
Database Type	Not specified	MySQL

**Note: Number and size of instances, size of AWS database, and number of Storm clusters vary depending on number of structures being monitored*

3.2 Qualitative Specifications

TABLE 4
High Level System Qualitative Specifications

Proposed Specifications	Demonstrated Specifications
Smooth real-time data transfer	Maximum Delay of 10 seconds
Low Power	✓
Low Maintenance	✓
Easy to use interface	✓

4. Design Approach and Details

4.1 Design Approach

The project involved designing an end-to-end prototype of a structural health monitoring system using a combination of hardware and software components and an IoT (Internet of Things) approach utilizing the cloud. The system was tested on a clothes dryer and detected motion, however the system was intended to be compatible with structures and buildings, small and large, as well as compatible with various types of readings. The changing vibrations of the clothes dryer simulated the potential change in motion of an actual structure.

4.1.1 Hardware

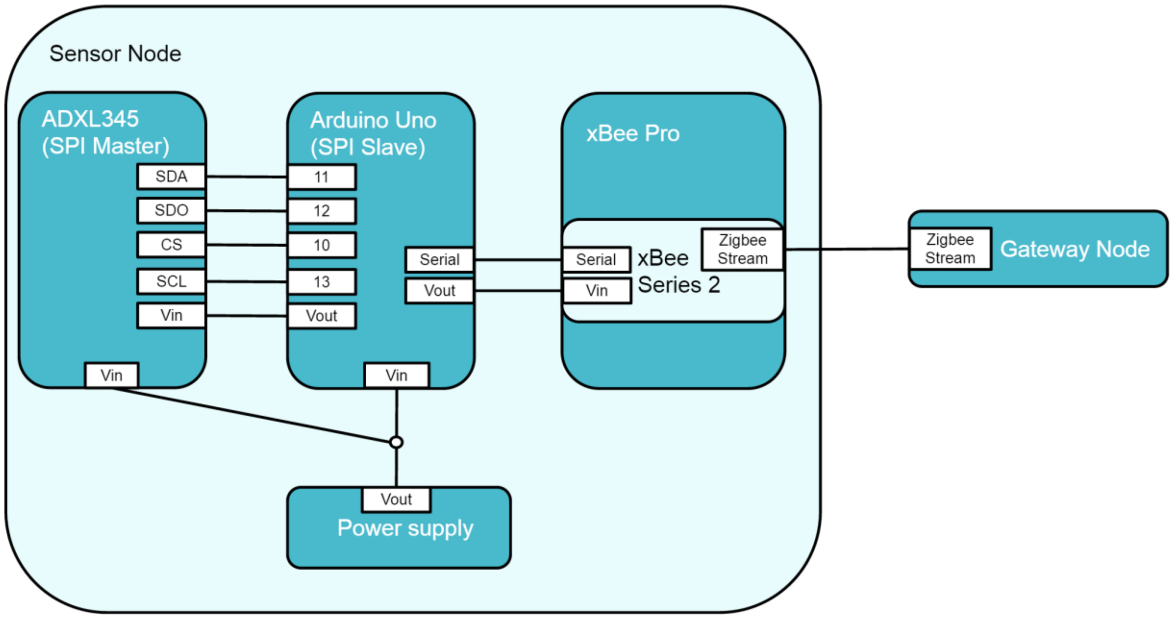


Figure 2. Internal details of a single sensor node with Zigbee connection to the gateway.

Each structure will have several sensor nodes, the prototype featuring three nodes. As shown in Figure 2, each node consisted of a 3-axis accelerometer sensor, an Arduino Uno microcontroller, and an xBee S2 module, all connected to a power source (battery or wired). Each structure contained several sensor nodes and one gateway Raspberry Pi device. Each node relayed sensor data to the gateway Raspberry Pi via a Zigbee (IEEE 802.15.4) wireless mesh network. Because Zigbee was used instead of 802.15.4 (found on xBee S1 chips), the nodes could relay data to the Raspberry Pi gateway through other sensors if there is not a direct line of sight to the gateway. The Zigbee modules (xBee S2s) were configured to operate in API mode instead of AT mode. API mode adds features like error detection through checksums and data queueing which removes the possibility of multiple sensors' data interleaving when it arrives at the gateway. These features required character escaping implementations at both the gateway and the sensor nodes, along with Zigbee packet size limitations [5].

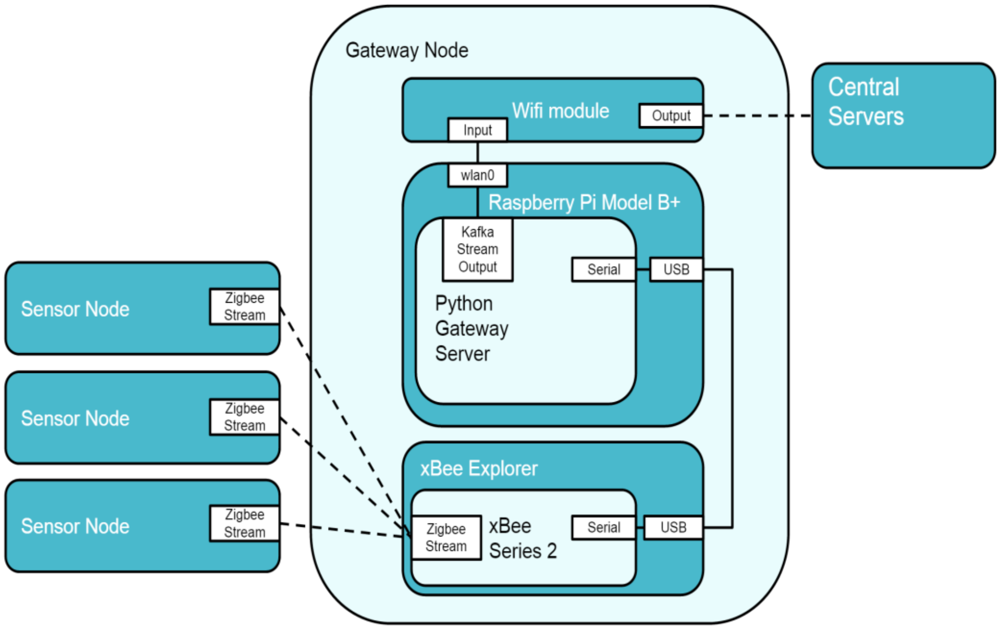


Figure 3. Internal details of the gateway node in the wireless mesh network.

The Raspberry Pi was configured to minimally manipulate the data before sending it to the cloud part of the system only so that it was formatted to be easily ingested by the analytics system. The team considered the possibilities of performing a small amount of filtering, analysis, or compression on the Raspberry Pi improved the system's efficiency. The Raspberry Pi might have required too much power or been unable to retrieve and send data fast enough if it was also required to analyze and simplify the data, especially if it were in a production system on a bridge with dozens of sensor nodes. It was therefore decided that all of the computational "heavy lifting" would be done by the Apache Storm component of the system. Additionally, the gateway was implemented with design-flexibility in mind so that if the sensor nodes were ever to switch to WiFi instead of Zigbee, the gateway program could easily be ported to another thread in the RTOS on the sensor nodes' microcontrollers. Therefore, Python dependencies were not too heavily relied upon in this component.

4.1.2 Cloud

The cloud portion of the design had four main components. First, an Amazon EC2 instance hosted Kafka, a tool used for real-time data streaming, to receive and send sensor data. Each structure's Raspberry Pi gateway hub was sending data to Kafka. Kafka served as the gateway of communication between structures' gateway hubs (Raspberry Pis) and the rest of the cloud infrastructure, therefore it was one of the highest priority items. Each structure created a Kafka topic which served as its individual pipeline to the analytics server. Kafka allowed the python application running on the Raspberry Pi to have a direct connection with the python application running analytics.

Another EC2 instance was deployed as an Apache Storm cluster which performed real-time analysis on the data retrieved from Kafka. Apache Storm, an open source tool, allowed the app to process large amounts of data in real time. The algorithm ran on incoming data in an Apache Storm topology made up of one spout and one bolt. The algorithms utilized data from the healthy state of the structure to determine the significance of current readings in relation to the previously analyzed healthy data. Readings outside of the healthy range resulted in the conclusion that the structure was in an unhealthy state.

The learning algorithm implemented in this design is critical to the accuracy of the system. Algorithm research and coding started at the beginning of the design process. The implementation of the Apache Storm topology enhanced the system further by making it quicker and more responsive to real-time events.

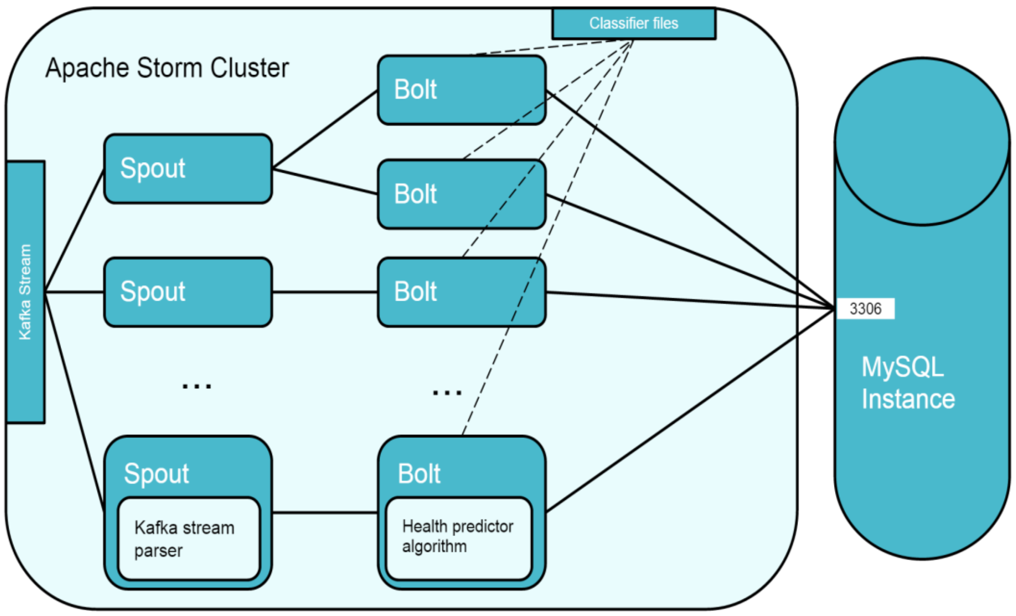


Figure 4. Internal details of Apache Storm topology.

A MySQL RDS hosted by Amazon Web Services stored the sensor analytics data, output by the Storm analytics. This data allows for the prototype to be extended to include long-term analysis. The persistence of the data means it was accessible to be analyzed further for long-term trends. It also makes it possible to update the algorithm to take into account newer data in the classification of current readings as healthy and unhealthy. The prototype did not include long-term analysis and the system did not update the algorithm with newer data.

Finally, a web application hosted on a third Amazon EC2 instance served as the user dashboard. The web application was developed with python using a Django framework.

4.1.3 Machine Learning Algorithm

Scikit-Learn is an open-sourced machine learning python library. It was chosen for this project due to its simplicity and flexibility as its functions are compatible with many different data types and dimensions. For this system, a support vector machine algorithm (SVM) was used.

SVMs are supervised machine learning algorithms, which analyze data to determine patterns. The scikit-learn module used was *svm.OneClassSVM* which is a novelty detection algorithm. First, a training data set, which is assumed to be healthy, is used to teach the algorithm what to expect for healthy readings. Then, real-time data is analyzed by the algorithm and classified as healthy or unhealthy based on how it relates to the training data set.

4.1.4 User Interface

Finally, the application had an intuitive user interface to display the analysis in a non-technical way. The user can view the readings from each sensor on the structure updating every second. The prototype featured a magnitude vs. time display of accelerometer data, as shown in Appendix A. It also displayed a general evaluation of the structure based on data output to the cloud-hosted database. The app allowed the user to view the sensor readings easily and in real-time.

4.1.5 Future Work Recommendations

The prototype was designed such that expansion is possible in four main ways:

- Implementation of a continuously learning algorithm would improve the long-term accuracy of the system. In addition, an improvement on the testing environment would allow for a more technical approach to the analytics. For example, details of an FFT approach are described below in *4.3.5 Sensor data and algorithms*.
- Many different types of sensors, such as the ones used in current health monitoring systems could be added to the design. The addition of temperature, wind, and/or pressure sensors could extend the accuracy of health monitoring and enable more complicated and intricate algorithms.
- Amazon Web Services, where the cloud resources were hosted, allows for resources to

be easily adjusted to customer needs. The inclusion of concepts such as load balancing would allow for this system to be more easily adapted to clients with smaller and bigger demands.

- Arduino Unos as the micro-controllers were ideal for prototyping as they could be easily configured to fit the system's needs. Other less expensive, lower powered devices could be used alternatively. One example is an MSP-430, an ultra low power TI microcontroller that is 1/3rd the cost of an Arduino Uno. Extra development work for the MSP-430 would have involved porting community-developed Zigbee and sensor libraries from Arduino to the MSP-430 pin configurations [7].
- If a lower-power microcontroller was selected, the sensor nodes could have run on solar power and rechargeable batteries for use cases such as deploying IoT-SHM on a bridge.

4.2 Codes and Standards

The IEEE 802.15.4 standard specified the physical layer and media access control for low power wireless mesh networks in which Zigbee standard is built on [6]. In this project, each sensor's readings were connected to an XBee module which was a part of a Zigbee mesh network comprised of all sensors on the same structure.

The Kafka protocol was developed by the Apache Software Foundation and simply uses six client request APIs. Kafka uses a binary protocol over TCP. The protocol defines all APIs as request response message pairs [7]. Kafka utilizes a publish-subscribe model in which producers publish data to a topic of their choosing and consumers subscribe to receive messages. For each topic, the Kafka cluster maintains a partitioned log which is an ordered, immutable sequence of messages that is continually appended to [8]. Kafka guarantees that messages sent by a producer will be stored in order and read in order by the consumer. For data transfer from the Raspberry Pi to the cloud services, this tool was used because of its unified, high-throughput, low latency platform for handling real-time data feeds.

4.3 Constraints, Alternatives, and Tradeoffs

4.3.1 Kafka vs. HTTP-Rest/Websockets/CoAP/MQTT communication between Raspberry Pi and the server

Given the need for constant readings from the accelerometer sensors, the sending of data was rapid and frequent. HTTP (hypertext transfer protocol) and Websockets required more overhead, were slower, and consumed more power than Message Queue Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP) [9], [10]. This system supported a quick transmission of data to immediately recognize changes in sensor readings. The CoAP protocol was better suited for state transfer of data while MQTT or Kafka were a better fit for live data transfer, which was necessary to monitor a structure in real-time [11]. Finally, like HTTP, Websockets and MQTT, Kafka supported security protocols as it ran over TCP [12]. In contrast, CoAP ran over UDP and was not secure [9]. Kafka's high-throughput and low-powered platform made it the best fit for this design. Kafka also managed the queueing of data without the system needing to implement a way to handle this.

4.3.2 Learning algorithm

Two possible algorithm types could have been used for this system. One would look for significant change in sensor readings or readings outside the threshold determined by the system designer. This approach simplifies calculations, but requires the designer to have knowledge of normal sensor readings in all healthy conditions for the specific building and is not easily adapted to a different building. Alternatively, a learning-algorithm could learn the healthy behavior of the given structure, and could be able to identify when the structure is behaving outside of expectations. This algorithm is complex, but is easily adapted to all structure and sensor types. A learning-algorithm was chosen for this project because of its flexibility and adaptability, in addition to it requiring no background knowledge from the designer.

4.3.3 Sensor data and algorithms

The accelerometer sensors in this design were chosen to reduce cost and power consumption. While the system is compatible with all data types, such as temperature or pressure, the prototype features only accelerometers. Accelerometers were chosen instead of other sensor types because motion would be easy to control for testing. Two data types were chosen, acceleration magnitude over time and the FFT output (magnitude for each frequency) over time. The chosen algorithm could be applied to either data-type, so the data-type that resulted in the lowest delay and fewest false health readings was chosen. First, the FFT was used, but the size of the FFT was limited by the memory of the Arduino which could only handle a maximum of 128 points. The resolution of the FFT is equal to the sampling frequency divided by the FFT size. Since the FFT size was limited to 128 and a high sampling frequency was needed to minimize the reading delay, the resolution, or minimum frequency that could be read, turned out to be higher than the resonant frequency of the controlled demo structure. Therefore, the acceleration magnitude in the time domain was chosen, but only for demo purposes. The FFT output would still better describe the internals of a large structure exposed to a lot of wind than acceleration magnitude vs. time would have.

4.3.4 Mesh-networking

There were three main communication alternatives for extracting data from the sensor nodes and sending the data to the servers. The first and simplest alternative was to pair each of the sensor nodes with the Raspberry Pi gateway via Bluetooth and communicate directly in a star-shaped mesh network. This design would take less development time than other alternatives, reducing the cost of the project, but the limitations are too severe to have proceeded with it. The Bluetooth protocol would allow the gateway to pair with a maximum of seven sensor nodes at a time, limiting the number of sensor nodes on a structure. Additionally, if one of the Bluetooth links disconnects, the system would not be able to reroute the failed sensor node to another link, making that element of the system fault intolerant. Since all of the sensor nodes would be directly linked to the gateway, the maximum distance of sensor nodes to the gateway would be limited by a Bluetooth link’s maximum range. This would make it impossible to apply the system to a structure as large as a bridge.

The second option was to put a WiFi module on each sensor node so that each sensor node communicates directly with the servers. In this case, there would be no need for a gateway server at each structure. Instead, each sensor node would be its own Kafka client and would send its data directly to the Kafka server. Aside from simplifying the system design and removing a point of failure by getting rid of the gateway server component, this technique would have also simplified advanced features like OTA software updates and anything else that would involve bidirectional communication.The biggest issue with WiFi came down to power usage. WiFi consumes four times as much power as Zigbee does, without adding any needed connection bandwidth [13]. Additionally, the WiFi nodes would not have been able to route data through each other, so many separate WiFi access points would have had to be purchased to give each sensor node enough bandwidth to communicate reliably with the Kafka server.

The third option, a Zigbee mesh network of sensor nodes with a gateway node, was the selected alternative. A Zigbee wireless mesh network allows nodes to relay messages through each other, forming dynamic minimum spanning trees to the gateway. Even though implementing a Zigbee network increased development time, this design almost completely eliminated all fault-tolerance issues described to be associated with the star-shaped Bluetooth mesh network and was much more cost-effective and power efficient than WiFi.

4.3.5 Structure type

This design was meant for structures that provide both power and internet access. The Raspberry Pi must be connected to a network in order to communicate with the cloud part of the system. In addition, this design depended on the ability of the Raspberry Pi to be connected to power at all times. This prototype was implemented for a building rather than a structure such as a bridge for these reasons.

4.3.6 NoSQL vs SQL databases

The schema for the data needed to store in this system fit well into simple narrow SQL tables. The tables are generic and adaptable to handle many buildings, sensors, and types of sensors. The need for a flexible data model, as provided in NoSQL databases [14], was not necessary for this project.

4.3.7 Database entry per reading or aggregate readings

The system sends to the database all of the necessary data from each reading as one entry. This method proved to reduce latency over time exponentially but with a small tradeoff: the webapp instance has to do some calculations when reading each entry. The alternative would be to write each data point from every reading individually (32 entries per reading). As demonstrated in Figure 5, this method caused a linear increase in latency which led to an extremely delayed response time for the system.

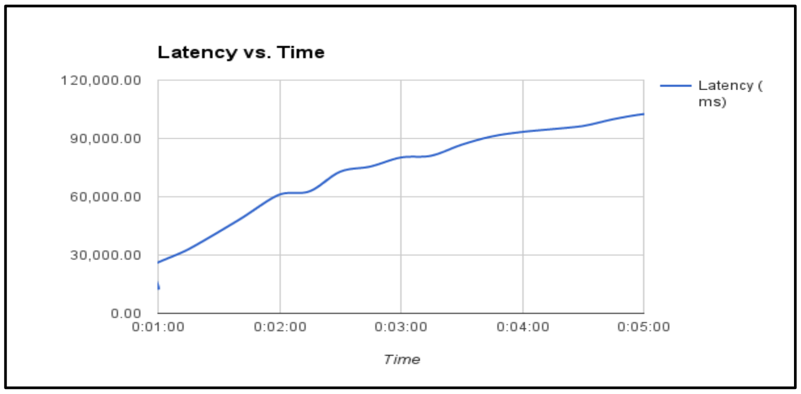
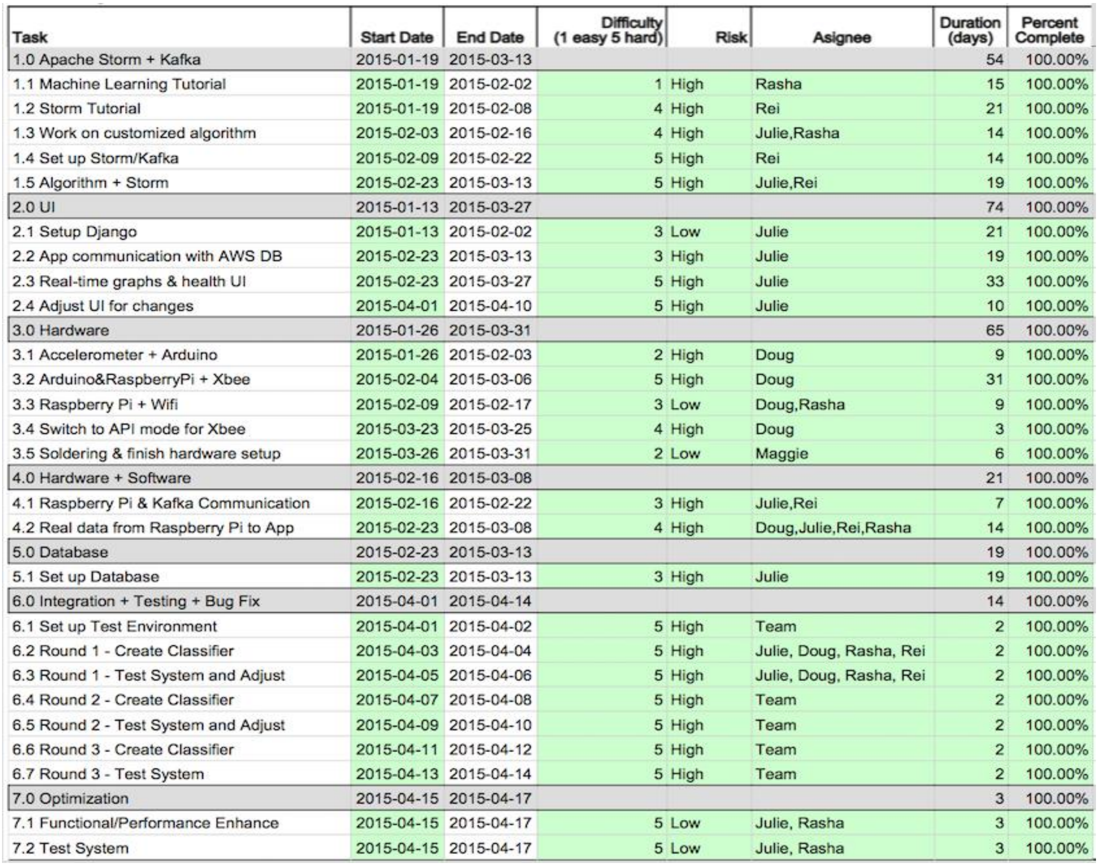


Figure 5. System latency when writing 32 entries per reading to database.

5. Schedule, Tasks, and Milestones

The Gantt chart in Appendix B shows Figure 6 in a graph form.



Task	Start Date	End Date	Difficulty (1 easy 5 hard)	Risk	Asignee	Duration (days)	Percent Complete
1.0 Apache Storm + Kafka	2015-01-19	2015-03-13				54	100.00%
1.1 Machine Learning Tutorial	2015-01-19	2015-02-02	1	High	Rasha	15	100.00%
1.2 Storm Tutorial	2015-01-19	2015-02-08	4	High	Rei	21	100.00%
1.3 Work on customized algorithm	2015-02-03	2015-02-16	4	High	Julie,Rasha	14	100.00%
1.4 Set up Storm/Kafka	2015-02-09	2015-02-22	5	High	Rei	14	100.00%
1.5 Algorithm + Storm	2015-02-23	2015-03-13	5	High	Julie,Rei	19	100.00%
2.0 UI	2015-01-13	2015-03-27				74	100.00%
2.1 Setup Django	2015-01-13	2015-02-02	3	Low	Julie	21	100.00%
2.2 App communication with AWS DB	2015-02-23	2015-03-13	3	High	Julie	19	100.00%
2.3 Real-time graphs & health UI	2015-02-23	2015-03-27	5	High	Julie	33	100.00%
2.4 Adjust UI for changes	2015-04-01	2015-04-10	5	High	Julie	10	100.00%
3.0 Hardware	2015-01-26	2015-03-31				65	100.00%
3.1 Accelerometer + Arduino	2015-01-26	2015-02-03	2	High	Doug	9	100.00%
3.2 Arduino&RaspberryPi + Xbee	2015-02-04	2015-03-06	5	High	Doug	31	100.00%
3.3 Raspberry Pi + Wifi	2015-02-09	2015-02-17	3	Low	Doug,Rasha	9	100.00%
3.4 Switch to API mode for Xbee	2015-03-23	2015-03-25	4	High	Doug	3	100.00%
3.5 Soldering & finish hardware setup	2015-03-26	2015-03-31	2	Low	Maggie	6	100.00%
4.0 Hardware + Software	2015-02-16	2015-03-08				21	100.00%
4.1 Raspberry Pi & Kafka Communication	2015-02-16	2015-02-22	3	High	Julie,Rei	7	100.00%
4.2 Real data from Raspberry Pi to App	2015-02-23	2015-03-08	4	High	Doug,Julie,Rei,Rasha	14	100.00%
5.0 Database	2015-02-23	2015-03-13				19	100.00%
5.1 Set up Database	2015-02-23	2015-03-13	3	High	Julie	19	100.00%
6.0 Integration + Testing + Bug Fix	2015-04-01	2015-04-14				14	100.00%
6.1 Set up Test Environment	2015-04-01	2015-04-02	5	High	Team	2	100.00%
6.2 Round 1 - Create Classifier	2015-04-03	2015-04-04	5	High	Julie, Doug, Rasha, Rei	2	100.00%
6.3 Round 1 - Test System and Adjust	2015-04-05	2015-04-06	5	High	Julie, Doug, Rasha, Rei	2	100.00%
6.4 Round 2 - Create Classifier	2015-04-07	2015-04-08	5	High	Team	2	100.00%
6.5 Round 2 - Test System and Adjust	2015-04-09	2015-04-10	5	High	Team	2	100.00%
6.6 Round 3 - Create Classifier	2015-04-11	2015-04-12	5	High	Team	2	100.00%
6.7 Round 3 - Test System	2015-04-13	2015-04-14	5	High	Team	2	100.00%
7.0 Optimization	2015-04-15	2015-04-17				3	100.00%
7.1 Functional/Performance Enhance	2015-04-15	2015-04-17	5	Low	Julie, Rasha	3	100.00%
7.2 Test System	2015-04-15	2015-04-17	5	Low	Julie, Rasha	3	100.00%

Figure 6. Tasks with assignments, difficulty, and risk level.

6. Final Project Demonstration

6.1 Test Setup

The system required two steps to demonstrate its capabilities. First, the system needed to be trained by collecting data with no outliers. Then, the trained system was tested to see if it can accurately determine the health of the system. This testing set-up used a clothes dryer as a structure. A clothes dryer was chosen because the motion of the machine was represented well in readings from accelerometers, the sensors chosen for this prototype. In addition, a dryer can simulate both a healthy state (off) and an unhealthy state (on). A computer running the web application displayed all live data sent from the system.

6.1.1 Training Phase

During training, the system created a classifier based on healthy data for the structure. This classifier was used by the algorithm during real-time analysis to determine characterize readings as healthy or unhealthy.

1. Sensors were placed on dryer.
2. Raspberry Pi received data from sensors and wrote to Kafka continuously.
3. Timed script retrieved data from Kafka for a given amount of time (4 hours) and wrote data points to a CSV file.
4. After timed script expired, the scikit-learn algorithm read data from CSV and created the classifier file.
5. The classifier file was added to the Storm topology.

6.1.2 Real-Time Analysis Phase

During real-time analysis, the system's ability to acquire and classify real-time data was demonstrated through the following steps:

6. Sensors were placed on a dryer.

7. Raspberry Pi received data from sensors and wrote to Kafka continuously.
8. Storm spout continually received and bolt analyzed data to determine health. Wrote results to database.
9. Current state of dryer (healthy) was displayed through the web application with current sensor readings.
10. Dryer was turned on.
11. Current state of dryer (unhealthy) was displayed through the web application with current sensor readings.

Figure 7 shows the classification of healthy versus unhealthy magnitudes by the scikit-learn algorithm for the dryer.

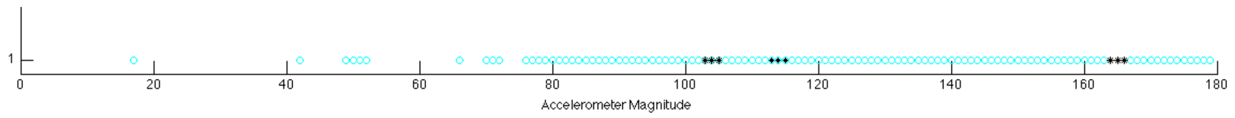


Figure 7. Scikit-Learn Healthy (Black) vs. Unhealthy (Cyan) Classification.

6.2 Project Specifications Demonstrated

- Real-time data transfer was reliable and fast
- System was low power
- The system was low maintenance
- The Web App was user-friendly
- Emergency response was not implemented to the system
- The Web App accurately displayed the health of the structure

6.2.1 Reliable and Fast Data Transfer

The system was demonstrated by switching a clothes dryer from off (healthy) to on (unhealthy). The delay between the dryer's state change and the web app's display of the state change was measured. This delay represents the total delay between the sensor sending the data to the gateway interface, the analysis of the data, and the web app receiving the information. For the dryer demonstration, the delay for the state change from healthy to unhealthy was two seconds and the delay for the state change from unhealthy to healthy was ten seconds.

6.2.2 Low Power System

In order to reduce the environmental costs of the project, the design leveraged low-power communication protocols. The sensor nodes, communicating via a Zigbee mesh network, use 75% less power for communicating than if they were each directly connected to WiFi.

6.2.4 User Friendly Web App

As shown by the figures in Appendix A, this system is supported by a user-friendly web app. The user's homepage lists all of their structures and displays the overall health of each structure.. If the user needs more information, they can select the structure and view individual sensors. The web app is customizable for the many data types the system supports.

6.2.5 No Emergency Response

The system did not have emergency response capabilities because it was trained to detect when behavior was not healthy, not when behavior was an emergency. In the future, the Web App would send push notifications to the manager to alert him or her that the structure is not healthy.

6.2.6 Web App Accurately Displays the Health of the Structure

In order for the system to be useful, it must always display accurate health information for the structure it is analyzing. Due to noise, every second the sensor readings over the last 10 seconds are averaged. The percent error for false unhealthy readings when healthy is determined to be 1.05% as there were 19 false unhealthy predictions for 1806 health predictions. The percent error for false healthy readings when unhealthy is determined to be 0% as there were 0 false unhealthy predictions for 1496 health predictions.

7. Marketing and Cost Analysis

7.1 Marketing Analysis

The IoT-SHM structural health monitoring system provides a wide variety of functionality for a low cost and should be used for general structural health monitoring on buildings. The target audience was governments and corporations who want low-cost and wide variability structural health monitoring for their buildings.

There were several other options already out there like those provided by Digitexx. Digitexx had recently developed a portable structural health monitoring system, called the PDAQ Premium, which costs $20,000 and offers 16 channels, 24-Bit Digitizer, embedded computer for local data viewing, local and real-time analysis, as well as 3 Client Software licenses for remote viewing from a laptop. In addition, the PDAQ Premium provided acceleration, wind speed and direction, displacement, strain, temperature, and pressure data [15].

One advantage of the IoT-SHM product is that it is less expensive since only an accelerometer sensor was used. Using self-sustainable sensor nodes and a fault-tolerant mesh network, the solution also requires minimal configuration and maintenance once installed. In addition, one of the biggest advantages of the IoT-SHM solution is that most computation for real-time and long-term analytics is done in the Cloud with ample computing resources, making the system more accurate by utilizing more data than if all computation were being done on local devices. The IoT-SHM platform can also be extended to home-automation since there is an API to access data from a structure's IoT-SHM. There is also a simple and extendable user interface that allows the user to see multiple structures' IoT-SHM systems that they subscribe to.

7.2 Cost Analysis

The prototype for this project consisted of three sensors, which were all assumed to be for the same building. As detailed below, the prototype would cost $259.85 to build, and would be sold for $800.00. If the client uses Amazon Web Services, they would have a cost of about $430 per year for a one structure, three sensor system.

The total cost of the overall prototype was $259.85 considering a system using three sensor nodes.

TABLE 5
Total Cost of Prototype

Part	Price Per Part	Quantity	Total Cost
Arduino Uno	$24.95	3	$74.85
Raspberry Pi Model B	$39.99	1	$39.99
5V 1A Switching AC Power Supply for the Raspberry Pi	$6.79	1	$6.79
ADXL345	$17.95	3	$53.85
Edimax EW-7811 150Mbps Wireless USB Adapter	$9.23	1	$9.23
XBee 1mW Trace Antenna - Series 2 (802.15.4)	$22.95	3	$68.85
8 Piece Set AA NiCd 600mAh 1.2V Rechargeable Battery	$6.29 [Amazon]	1	$6.29
Packaging	$0.00 (3D printer)	1	$0.00
Total Cost of Prototype			$259.85

Assuming a typical starting engineer's salary of $30 per hour, the following development costs were determined for the IoT-SHM product [16]. The most time consuming tasks consisted of the independent research and the sensor and microcontroller programming since those were the most demanding requirements.

TABLE 6
Total Development Labor Costs

Activity	Duration (Hours)	Cost
Team Meetings	20	$600.00
Report Preparation	30	$900.00
Independent Research	60	$1800.00
Design	30	$900.00
Assembly	10	$300.00
Programming Microcontrollers and Sensors	60	$1800.00
Debugging	30	$900.00
Presentation Preparation	30	$900.00
Total Labor Cost		$8100.00

Adding in fringe benefits and the overhead which account for 30% and 150% of the total labor cost, respectively. Table 7 summarizes the determined development costs.

TABLE 7
Total Development Cost

Component	Cost
Prototype	$259.85
Development Labor	$8100.00
Fringe Benefits	$2430.00
Subtotal	$10789.85
Overhead	$16184.78
Total Development Cost	$26974.63

If approximately 2000 systems were sold in five years, the development cost per system would be approximately $13.49. With a discount of approximately 25%, the hardware cost would drop to $194.89. Since IoT-SHM was a small business, a smaller sales percentage (8%) would be employed in the advertising of these devices [17]. Assuming that a salesperson was paid $10 an hour for assembly and testing and that assembling and testing one part took half an hour each since the design was very simple to put together, then the amount paid for assembly and testing per part would be $5.00 each. The overall profit would then be $202.79 per part at a cost of $800 per product with a 5-year revenue of $405580.

TABLE 8
5-Year IoT-SHM Production

Component	Cost
Hardware	$194.89
Testing	$5.00
Assembling	$5.00
Total Labor	$10.00
Fringe Benefits	$3.00
Subtotal	$207.89
Overhead	$311.83
Sales	$64.00
Amortized Development Cost	$13.49
Profit	$202.79
Price Per Unit	$800.00

Actual cost for the client would depend on the cloud computing service plan chosen by the client to fit their needs. For our project, Amazon Web Services was chosen since it is fairly priced and easy to use. Our group utilized a medium and an extra large instance. The average cost of a medium instance was 0.0081 dollars per hour and the average cost of an extra large instance was 0.0410 dollars per hour. Therefore, a client who wished to use three sensors on a single building would pay 1.174 dollars per hour or 430 dollars per year for the web services necessary to maintain this system.

In addition to AWS, Amazon Relational Database Service (RDS) for MySQL was used in the prototype to set up a MySQL database in the Cloud. The database stores the time, magnitude, and health of the individual sensors. Again, Amazon RDS has various plans depending on the storage needs of the user. This prototype used the most cost-effective service, the single AZ deployment, which costs 0.12267 $ per GB used each month. The actual cost to the client would depend on if they are interested in long-term health trends as the GB usage per month can be reduced by deleting unneeded data.

8. Conclusion

8.1 Current Status

8.1.1 Completed Development

Currently, each major component of the system has been implemented enough to generate structural health data (based on only accelerometer sensor data) and a nearly production level web app interface to display a live feed of a structure's health and sensor reads. Currently, structural health detection is based on novelty detection of raw 3-axis accelerometer sensor reads. Each software component (Arduino microcontrollers, Raspberry Pi gateway, Kafka server, Apache Storm, MySQL database, and the web application) also has a separate branch for handling FFT outputs instead of raw sensor reads so that feature can easily be turned back on if a more realistic demo was set up and the FFT output was deemed more useful. The novelty detection is generated by the Python scikit learn library and is executed by an optimized Apache Storm topology so that the system is able to determine the health in real time. The Zigbee mesh network is completely configured in API mode with bidirectional communication capabilities between the gateway and each microcontroller.

8.2 Lessons Learned

Regardless of the electrical and computer engineering challenges faced in the project, having real domain expertise would have served invaluable for designing a system that is working to solve a problem for a topic as complex as structural health monitoring. Now that the hardware and software platforms have been laid out and optimal components for each step have been decided on, this project could take a new interdisciplinary direction for future study.

8.3 Future Expansions

8.3.1 Roadmap for a Low-Maintenance, Sustainable System

In order to reduce the environmental costs of the project, the design leveraged low-power communication protocols. The sensor nodes, communicating via a Zigbee mesh network, use 75% less power for communicating than if they were each directly connected to WiFi. Additionally, since WiFi access is not necessary, the sensor nodes can be placed on the outer walls of a structure where they can be powered by solar solar energy. In order to make implementing solar recharging more feasible, the Arduino Uno microcontroller would have to be swapped out for a lower power device, like a TI MSP430 [18]. Once solar power is in place, the goal would be a zero carbon footprint outside of the cloud services.

In order to ensure zero physical sensor node maintenance, OTA software updates for the gateway and sensor nodes, along with sensor fault detection would have to be put in place. Additionally, ensuring that the sensor nodes are packaged correctly so that they are as heat tolerant as possible would be required.

8.3.2 Increasing Damage Detection Accuracy

Implementation of a continuously learning algorithm would improve the long-term accuracy of the system. Sensors like thermometers, strain gages, and piezoelectric patches could be added to broaden the data being pulled from the structure. This could give the system a more realistic outlook on the actual health of a structure. A piezoelectric sensor was used in the system in the beginning but due to time constraints and a lack of documentation on the chosen sensor to get an accurate sampling rate, the team was unable to implement it in the final demonstrated prototype.

8.3.3 Long Term Health Feature

If the team had domain expertise in structural health monitoring, a more long-term algorithm outside of vibration novelty detection could have improved the viability of the system. With this new feature, there would be a new screen on the web dashboard that shows long term trends of different parameters that could help indicate structural health.

8.4.4 Web Scalability

Currently, the web application is running on a single Amazon Web Services (AWS) EC2 instance. Since AWS has a nearly endless supply of instances, the system could able to be easily adjusted to customer needs. If instances hosting the web server application were to be added, load balancing would allow for this system to be more easily adapted to clients with smaller and bigger demands. Additionally, the database holding the sensor data could be split among different instances and old historical data could be compressed.

9. References

[1] “PDAQ Basic” [Online]. Available: http://www.digitexx.com/portable_structural_monitoring_products/pdaq-basic/ . [Accessed Oct. 20, 2014].

[*2*] Accelerometer,” *SensorWiki.org*, Apr. 19, 2013. [Online]. Available: http://www.sensorwiki.org/doku.php/sensors/accelerometer. [Accessed Nov. 18, 2014].

[3] Amazon Web Services (2014). “What is Cloud Computing?” [Online]. Available: http://aws.amazon.com/what-is-cloud-computing/ [Accessed Nov. 26, 2014].

[4] A. Bahga and C. Madisetti, *Internet of Things: A Hands-On Approach*, 2014.

[5] Digi International, “XBee®/XBee-PRO® ZB RF Modules,” xBee S2 datasheet, Mar. 2012.

[6] G. Sanchez, F. Garcia-Sanchez, D. Rodenas-Herraiz. "On the Synchronization of IEEE 802.15.5 Wireless Mesh Sensor Networks: Shortcomings and Improvements." *EURASIP Journal on Wireless Communications and Networking* 2012.1 (2012): 198. Web.

[7] J. Kreps, "A Guide To The Kafka Protocol" [Online]. Available: https://cwiki.apache.org/confluence/display/KAFKA/A+Guide+To+The+Kafka+Protocol#AGuideToTheKafkaProtocol-Network . [Accessed Apr. 12, 2015].

[8] “Apache Kafka: A high-throughput distributed messaging system” [Online] . Available: http://kafka.apache.org/documentation.html . [Accesses Apr. 12, 2015].

[9] P. Patierno, “MQTT & IoT protocols comparison,” presented at the Microsoft Embedded Conf., Naples, Italy, 2012. [Online]. Available: http://www.slideshare.net/paolopat/mqtt-iot-protocols-comparison

[10] S. Nicholas, “Power Profiling: HTTPS LongPolling vs. MQTT with SSL, on Android”, May 2012. [Online]. Available: http://stephendnicholas.com/archives/1217 [Accessed Oct. 20, 2014]

[11] T. Jaffey. (2014). “MQTT and CoAP, IoT Protocols” [Online]. Available: http://eclipse.org/community/eclipse_newsletter/2014/february/article2.php [Accessed Nov. 25, 2014]

[12] MQ Telemetry Transport (MQTT) V3.1 Protocol Specification, [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html. [Accessed Oct. 15, 2014]

[13] “Difference Between Zigbee and WiFi Technologies” [Online]. Available: http://www.engineersgarage.com/contribution/zigbee-vs-wi-fi. [Accessed 2/27/2015]

[14] O.S. Tezer, “Understanding SQL and NoSQL Databases And Different Database Models,” *digitalocean.com*, 21 Feb 2014. [Online]. Available: https://www.digitalocean.com/community/tutorials/understanding-sql-and-nosql-databases-and-different-database-models [Accessed Nov. 26, 2014]

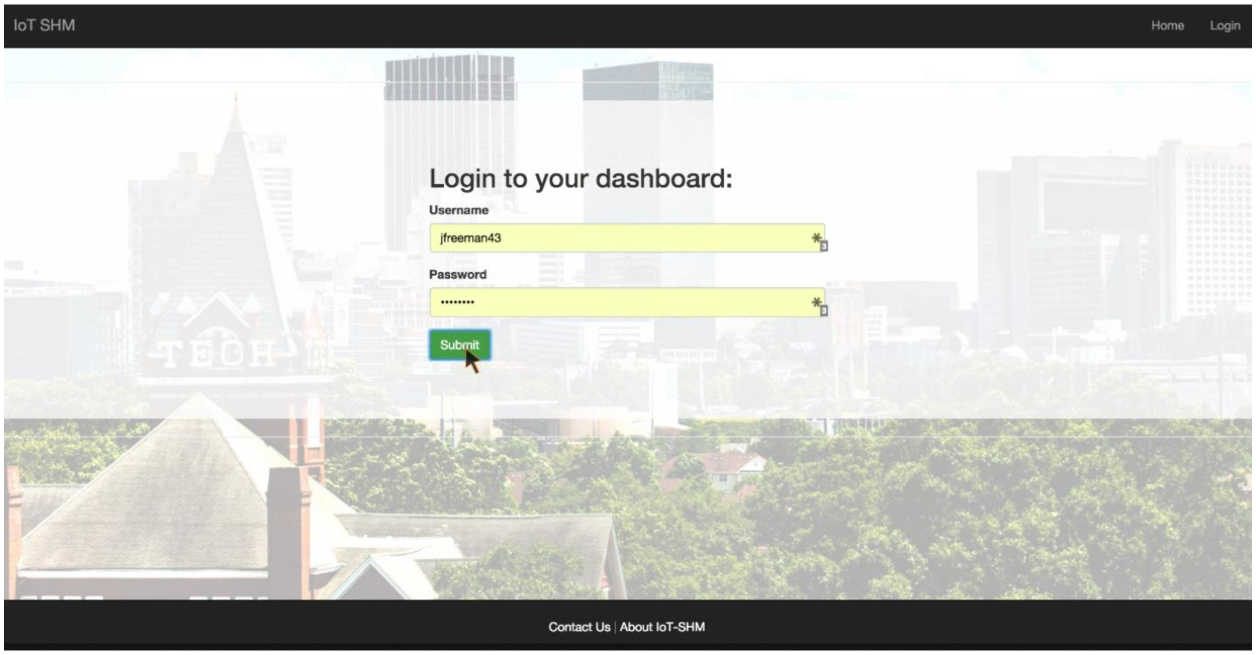
[15] Digitexx: Data Systems, Inc., *PDAQ (Portable Data Acquisition Systems) Product Family*.

[16] PayScale: Human Capital, “Bachelor of Science (BS / BSc), Electrical Engineering (EE) Degree Hourly Rate,” *payscale.com*, Nov. 15, 2014. [Online]. Available: http://www.payscale.com/research/US/Degree=Bachelor_of_Science_%28BS_%2f_BSc%29%2c_Electrical_Engineering_%28EE%29/Hourly_Rate. [Accessed Nov. 18, 2014].

[17] G. Boykin, "What Percentage of Gross Revenue Should Be Used For Marketing and Advertising?" [Online]. Available: http://smallbusiness.chron.com/percentage-gross-revenue-should-used-marketing-advertising-55928.html. [Accessed Nov. 18, 2014].

[18] A. Allan, “Which Board is Right for Me?” [Online]. Available: http://makezine.com/magazine/make-36-boards/which-board-is-right-for-me/. [Accessed 2/26/2015].

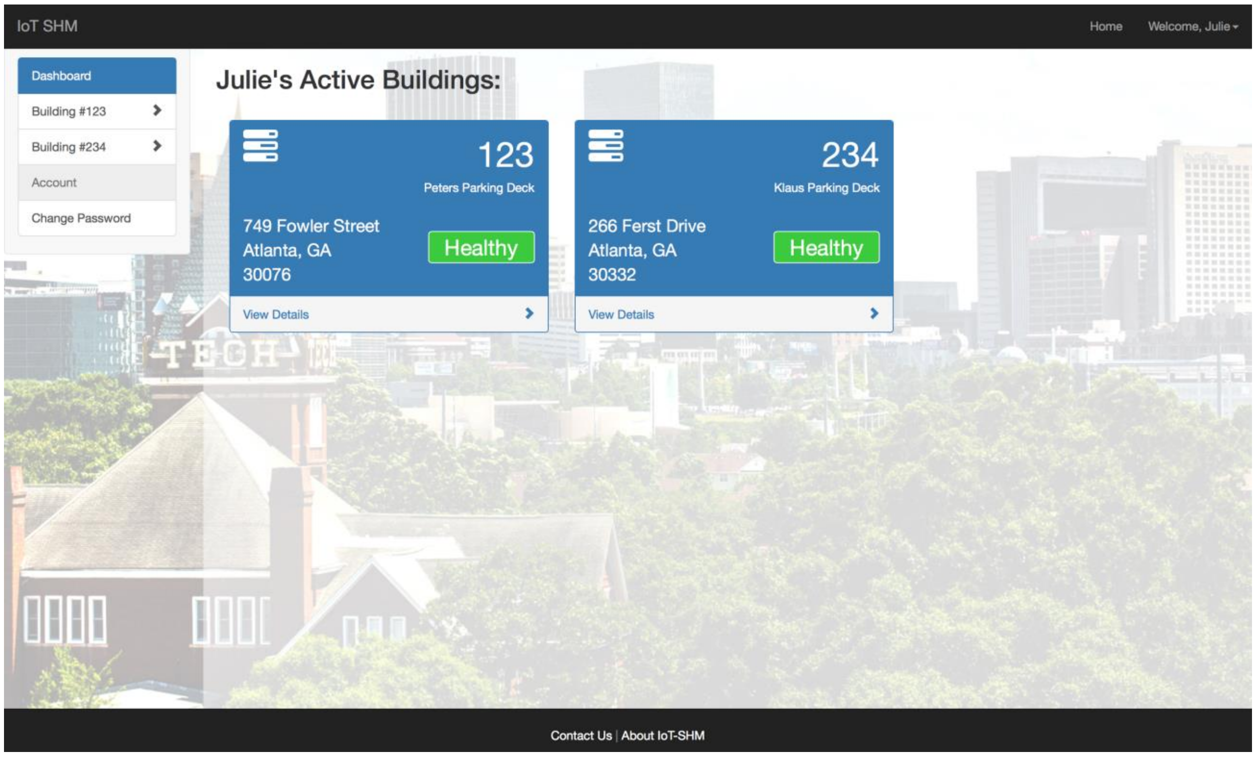
Appendix A - UI Screenshots



Login to your dashboard:

Username: jfreeman43

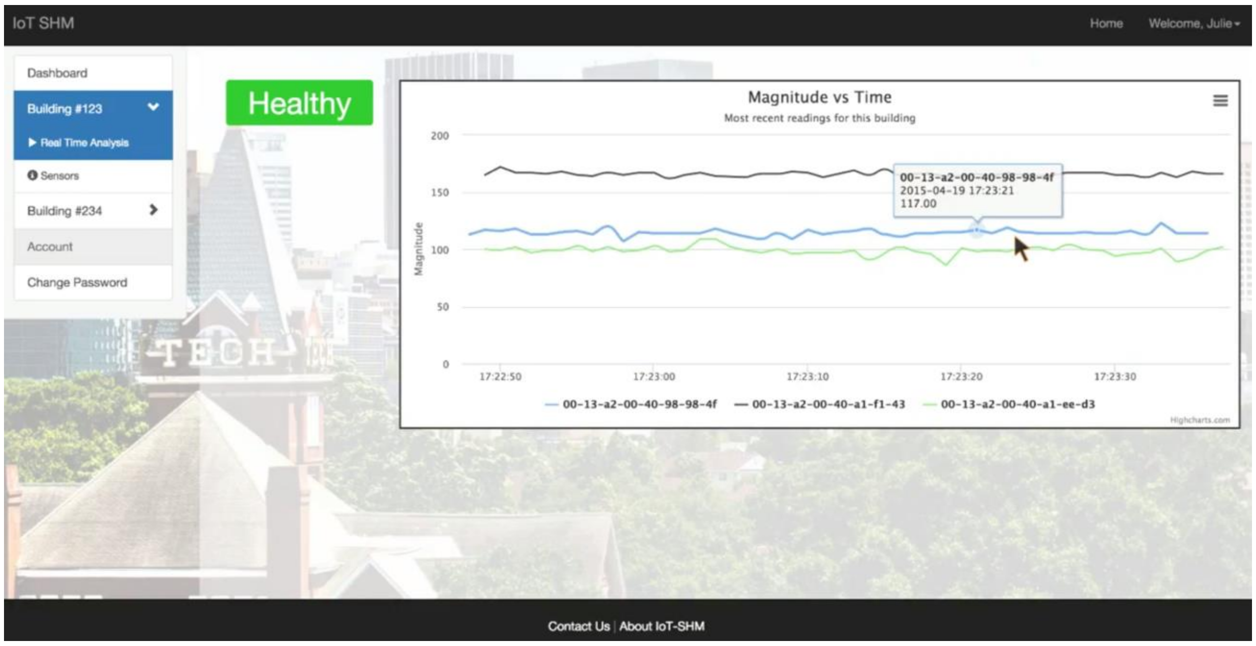
Password: ••••••••



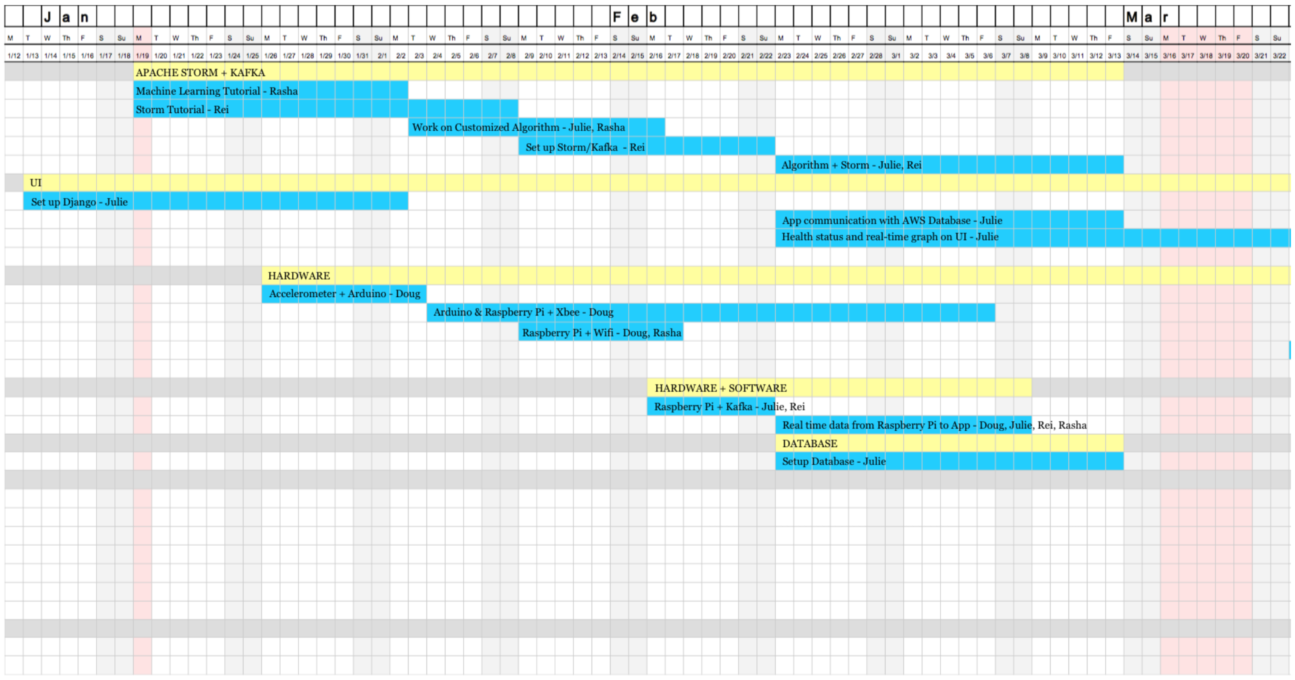
Julie's Active Buildings:

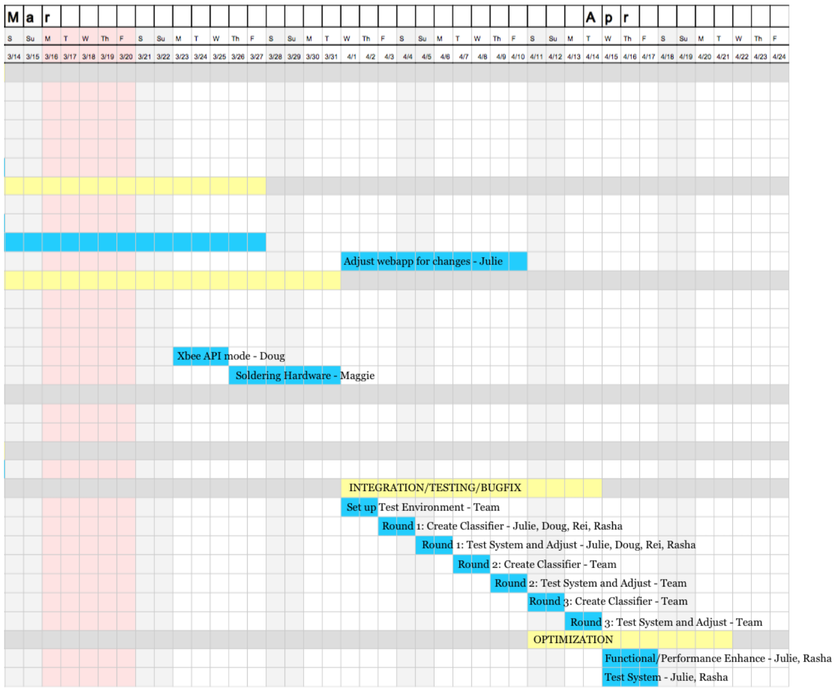
123
Peters Parking Deck
749 Fowler Street
Atlanta, GA
30076
Healthy

234
Klaus Parking Deck
266 Ferst Drive
Atlanta, GA
30332
Healthy


IoT SHM
Home
Welcome, Julie
Dashboard
Building #123
Real Time Analysis
Sensors
Building #234
Account
Change Password
Healthy
Magnitude vs Time
Most recent readings for this building
Magnitude
00-13-a2-00-40-98-98-4f
2015-04-19 17:23:21
117.00
00-13-a2-00-40-98-98-4f
00-13-a2-00-40-a1-f1-43
00-13-a2-00-40-a1-ee-d3
Highcharts.com
Contact Us | About IoT-SHM

Appendix B - Project Gantt Chart


Jan
Feb
Mar
APACHE STORM + KAFKA
Machine Learning Tutorial - Rasha
Storm Tutorial - Rei
Work on Customized Algorithm - Julie, Rasha
Set up Storm/Kafka - Rei
Algorithm + Storm - Julie, Rei
UI
Set up Django - Julie
App communication with AWS Database - Julie
Health status and real-time graph on UI - Julie
HARDWARE
Accelerometer + Arduino - Doug
Arduino & Raspberry Pi + Xbee - Doug
Raspberry Pi + Wifi - Doug, Rasha
HARDWARE + SOFTWARE
Raspberry Pi + Kafka - Julie, Rei
Real time data from Raspberry Pi to App - Doug, Julie, Rei, Rasha
DATABASE
Setup Database - Julie


Mar
Apr
Adjust webapp for changes - Julie
Xbee API mode - Doug
Soldering Hardware - Maggie
INTEGRATION/TESTING/BUGFIX
Set up Test Environment - Team
Round 1: Create Classifier - Julie, Doug, Rei, Rasha
Round 1: Test System and Adjust - Julie, Doug, Rei, Rasha
Round 2: Create Classifier - Team
Round 2: Test System and Adjust - Team
Round 3: Create Classifier - Team
Round 3: Test System and Adjust - Team
OPTIMIZATION
Functional/Performance Enhance - Julie, Rasha
Test System - Julie, Rasha