

Bachelor's thesis

Degree Programme in Information Technology

Internet Technology

2017

Tam Nguyen Truong Thanh

MONITORING ENVIRONMENTAL VARIABLES USING SENSORTAG



BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2017 | 42

Tero Virtanen

Tam Nguyen Truong Thanh

MONITORING ENVIRONMENTAL VARIABLES USING SENSORTAG

The purpose of this thesis was to create a system consisting of a multi-functional sensor and a web-based application in order to monitor environmental variables, such as temperature, humidity, light, density and so on. With an intention of using the sensor to track object movements, a gyroscope sensor was additionally created. The application, called sensorApp, can be seen as a full-stack web developing example which takes advantage of such technologies as NodeJS, MongoDB, jQuery and Socket.io.

The back-end part of this project is a NodeJS web server communicating with the sensor device in order to retrieve information then forward to to a MongoDB database server for storing. On the other hand, for the front-end part, the data received from the sensor is also transferred to a web-interface to present and monitor in real-time with the help of WebSocket's library named Socket.io. In addition, the HighCharts library is used to create graphs to visualize all the data from the sensor or from the database server.

The result of the thesis was the "sensorApp" app. The app is capable of presenting data from the sensors and, especially, is packed with database extraction feature which is not supported by other services. There were few obstacles in this project, such as hardware faulty, overloaded data transmission and security methods. In addition, limitations in technology were also addressed, for instance power consumption and gyroscope accuracy.

KEYWORDS:

Node.js, MongoDB, JavaScript, jQuery, Sensor, Web, Server.

CONTENTS

LIST OF ABBREVIATIONS (OR) SYMBOLS	6
1 INTRODUCTION	6
2 TECHNOLOGIES	8
2.1 SensorTag	8
2.2 Bluetooth Low Energy	8
2.3 NodeJS	9
2.4 NodeJS Modules	9
3 PROCESS	12
4 IMPLEMENTATION	14
4.1 NodeJS and modules	14
4.2 JavaScript Libraries	15
4.3 Wire up	16
4.4 Setting up connection	17
4.5 Data Flow	20
4.6 User interface	23
4.7 Database	25
5 PERFORMANCE	27
6 ASSESSMENT	29
6.1 Obstacles	29
6.2 Potential Future Improvement	29
6.3 Limitations	30
7 CONCLUSION	34
REFERENCES	35

APPENDICES

- Appendix 1. Application Tree
- Appendix 2. Sensortag module code

EQUATIONS

Equation 1. Differences when sensor changing position.	32
--------------------------------------------------------	----

FIGURES

Figure 1. Data transfer from UI to server via “custom” socket.	21
Figure 2. Socket “valuesOut” structure.	21
Figure 3. Socket “signal”.	22
Figure 4. Records are sent to database.	22

PICTURES

Picture 1. Concept of a monitoring system.	6
Picture 2. SensorTag CC2650.	8
Picture 3. The implementation structure.	12
Picture 4. Discovery mode.	17
Picture 5. Broadcasting connectivity signal.	18
Picture 6. Sensor information recognized by node server.	18
Picture 7. Sensor panel in dashboard.	19
Picture 8. Responses from server when connected to SensorTag.	20
Picture 9. Dashboard interface.	23
Picture 10. Show records panel.	24
Picture 11. Visualized graph of data.	24
Picture 12. Exporting feature of HighCharts.	25
Picture 13. Structure of a key in MongoDB database.	26
Picture 14. A point transferred from server to graphs.	27
Picture 15. How data is handled upon requests.	28
Picture 16. Light density presentation.	30
Picture 17. Gyroscope presentation.	32

TABLES

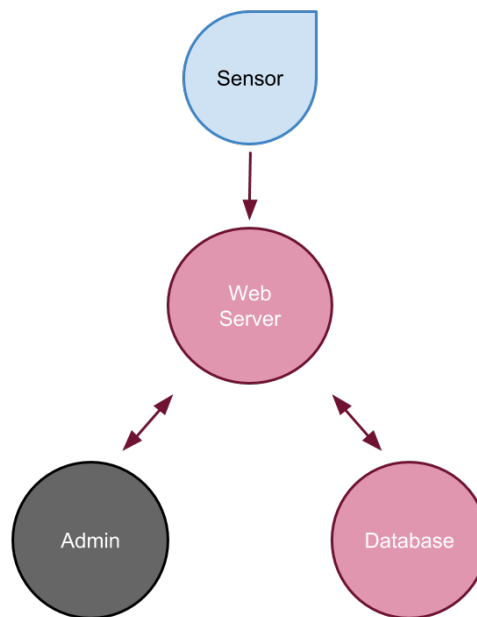
Table 1. JSON format.	11
Table 2. NPM installing syntax.	14
Table 3. Create project with Express Generator.	14
Table 4. Sensortag module install command.	15
Table 5. Link to “socket.io” library in HTML.	15
Table 6. Get “socket.io” functions in JavaScript.	15
Table 7. Link to your mLab database.	16
Table 8. Link to HighCharts Library in HTML.	16
Table 9. Start NodeJS server command.	17
Table 10. Power consumption.	31

LIST OF ABBREVIATIONS (OR) SYMBOLS

API	Application Program Interface
BLE	Bluetooth Low Energy
DOM	Document Object Model
HTML	Hyper Text Markup Language
IoT	Internet of Things
JSON	JavaScript Object Notation
MVC	Model View Controller
RDBMS	Relational Database Management System
UI	User Interface
XML	Extensible Markup Language

1 INTRODUCTION

Monitoring a server plays an essential role in an administrator's routine. As the network grows, keeping track of all the activities is becoming more and more complicated. Moreover, the performance of servers does not only depend on the hardware or software alone but also on other factors such as temperature or humidity of surrounding environment. Especially, overheating is one of the main reasons to cause increasing latency in networks. Therefore, the need of a surrounding monitor system which can be supervised remotely seems to be necessary.



Picture 1. Concept of a monitoring system.

As presented in Picture 1 above, the centralized system is in charge of retrieving data from any sensor built-in devices. Sensor data consists of temperature, humidity, light density, magnetic level, etc. The raw data is transferred to the system where it will be converted into human readable values. While being processed, the data is also captured by the database server then saved in form of records. The advantage of the database server here is to keep track of all measurements during the whole operation. The database records are also used as a future reference or to analyze trending of sensor values. There are many forms of presenting data for administrator. Since the requirement is to monitor remotely with any devices available, a website is considered as the best choice.

This thesis demonstrates the process of establishing the monitoring system. With the web-based application being a core of the system, the application does not provide only a graphical interface that suits every device from computer to mobile phone but also a real-time reporting functionality.

The thesis starts with introducing all the technologies used in the project. Chapter Process explains why the selected technologies are suitable for the project. Chapter Implementation describes how the project was implemented. Chapter Performance reports on how the performance was validated. The final chapter discusses obstacles and limitations in this project.

2 TECHNOLOGIES

2.1 SensorTag

SimpleLink™ Bluetooth Smart®/Multi-Standard SensorTag from Texas Instrument is an expandable sensor-based Bluetooth device. There are many versions but the one we use in this project particularly is CC2650 SensorTag. Basically, this version has 10 sensors built-in such as support for light, digital microphone, magnetic sensor, humidity, pressure, accelerometer, gyroscope, magnetometer, object temperature, and ambient temperature[1] on the same circuit and operates at a low energy level. In terms of expandability, more sensors could be added with DevPack in order to fit every need. Regarding power consumption, according to Texas Instrument, with an interval of 1 second, it can runs for at least 1 year straight. Moreover, it also provides an option to transfer measurements to the Cloud via an app for both iOS and Android platforms. In addition, iBeacon, ZigBee®/6LoWPAN technology are also supported by the sensor.



Picture 2. SensorTag CC2650.

Development kit (DevPack) [2] is an optional circuit which provides USB interface for SensorTag to enable the physical connection to computer. Updating firmware and making modification to the sensor are the main roles of the kit. On the other hand, it also helps the sensor to use external power supply when running out of battery.

2.2 Bluetooth Low Energy

Bluetooth Low Energy (BLE) or Bluetooth Smart is a latest version of Bluetooth Technology [3]. In order to accommodate Internet of Things (IoT) trend, BLE now consumes less power than previous versions so it can keep operating for long time with only a coin-cell battery. Regarding security, BLE can enhance security ability such as fully compatible with 128-bit AES encryption. Moreover, with a wide availability along with mentioned features above, BLE is gradually making its way to more and more IoT projects.

2.3 NodeJS

To accomplish a huge stream of data constantly transferring to the server, one of the best implementation in the market at the moment is NodeJS server.

Node.js / NodeJS is an event-driven environment. It is a library was invented by Ryan Dahl in 2009. After 45-minute talk at JSConf, Ryan introduced NodeJS which changed JavaScript forever[4]. With an intention of expanding JavaScript beyond just a web browser language, NodeJS now becomes one of the most powerful servers in the market. A NodeJS server is capable of handling a vast amount of connections at once. Despite the fact that it is a single-thread server, it provides a decent performance as opposed to other multi-thread servers. Concurrent operations is smoothly executed with the help of V8 engine which is an engine from Google built to improve the performance of JavaScript. There are a excessive number of APIs and modules available for NodeJS out of the box. Thousands and thousands of modules are developed and contributed by community up to now makes NodeJS adapt itself to many projects or is widely used by large companies.

2.4 NodeJS Modules

NodeJS has a modular-oriented structure. Modules are blocks of code in certain language which are built to accomplish certain tasks. Their functionalities could be standalone or combined with each other in order to achieve much sophisticated operations.

Sensortag

Sensortag module for NodeJS was created by Sandeep Mistry in 2013 [5]. It is an API made specifically for Texas Instruments SensorTags, namely TI CC26xx series. This module is one of the most important pieces of this app. It provides functions to establish a connection between the sensor and the server as well as allows developers display extracted detail information of the device for example serial numbers, type, name or even firmware revisions. Apart from connectivity, the API also enables us to take control over built-in sensors in just a few lines of code. In other words, from enabling a customized sensor to exporting data in order to process in further stages, all of those tasks can be done easily without any knowledge of low level programming language. The comprehensive list of provided categorized functions as follows:

- Discovery
- Connect / Disconnect device
- Device Information
- Sensors
 - Accelerometer
 - IR Temperature
 - Humidity
 - Magnetometer
 - Barometric Pressure
 - Gyroscope
 - Luxometer (CC2650 only)
 - IO
- Simple Key

Socket.io

Since web application is essentially mainstream, data which traverses across networks demands a high response rate in order to maintain user experience. In modern web applications, user inputs are frequently processed in real-time, then exchanged back and forth between back-end and front-end sections. That is when a definition of WebSocket comes along. Basically, WebSocket [6] is intentionally a protocol which enables us to set up a communication channel for message exchanging from a server to clients or vice versa.

Regarding compatibility issues between modern browsers and older ones, socket.io was invented to maintain a consistency for using WebSocket on any browsers without adjustments. As a matter of fact, socket.io grows incredibly fast and is implemented in the majority of web applications along side with NodeJS.

jQuery

jQuery was created as a JavaScript library. Technically, everything can be done with JavaScript, which now is even faster and simpler with jQuery. Moreover, user interaction is now handled in a more straight-forward way. There are various types of activity happening on-site, for example, DOM manipulation, event schedule which now can accomplish even more rapidly with jQuery [7]. In addition, pre-created snippets for animations are available right out of the box for developers as well. All in all, jQuery is widely used in vast numbers of projects in any scale and become one of the most popular JavaScript libraries without a doubt.

MongoDB

A database essentially is a must-have in a web application. Since the beginning, RDMS has taken a lead in the web development industry. As a result of the dominance of large-scale web applications, the NoSQL database has becoming strong. Because of it numerous features such as scalable, adaptive to changes, supporting JSON-like format, NoSQL has received significant attention from start-ups to large scale enterprises. One of the most popular databases in this category is MongoDB [8]. Apart from characteristic inheritance of non-relational databases, MongoDB's most highlighting feature is a JSON-like database. JSON format has been widely used to transfer data between clients and servers.

Table 1. JSON format.

```
{
    'key' : value ,
    'Name' : "John",
    'ID' : 1
}
```

The example above is a simple declaration of a JSON-format variable. As presented in a pair of key and value, JSON has a more flexible structure than its former, XML. In addition, a value can be stored in form of a JSON as well.

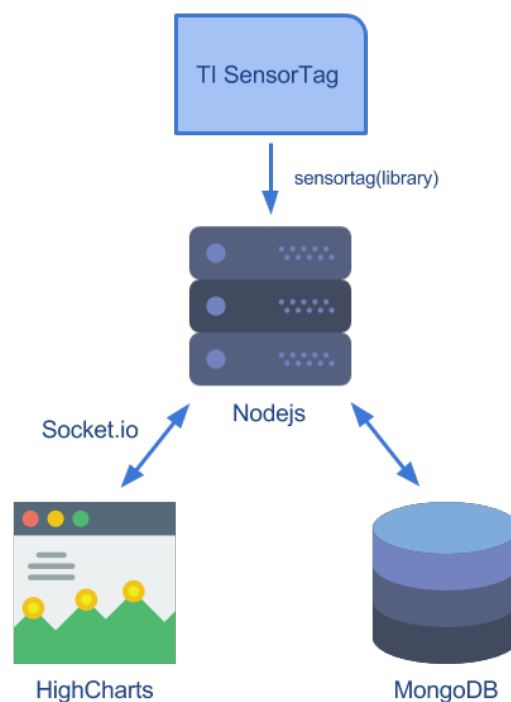
With the use of JSON scheme, data from server now can be directly stored in MongoDB without making any conversions. In addition, complicated table constraints of traditional database are completely eliminated. In terms of application scaling, it is extremely unsophisticated to append more records or attributes as the app grows. The mentioned factors make MongoDB suitable for any criteria of this project especially in mobile generation where JSON can be seen as a new standard for data type.

3 PROCESS

After carrying out some research to decide which technology is suitable for this project, a list of recent ones were selected as follows: NodeJS, MongoDB, jQuery, HighCharts, Socket.io.

The main reason for choosing NodeJS in the first place is its high performance which is ideal for real-time web applications. Apart from that, thousands of plugins are developed and available to accommodate any requirements. Especially, sensortag is the one library that can enable developers to open up the potential of Texas Instrument sensors. With that foundation, the system already has a web server with built-in functionality to connect and communicate with the sensor device.

In terms of databases, MongoDB was chosen since it is easy to incorporate with NodeJS and recommended the most in web development community. Measurements from the sensor are stored in MongoDB while they are presented on the website simultaneously. The database comes in handy when it comes to review the trending analysis of values from the sensor in the future. Moreover, the ability to interact with JSON format also is taken into account.



Picture 3. The implementation structure.

With the convenience of JSON structure, HighCharts [9], a jQuery library can process data from database transparently then display it in an appealing way. The advantage of HighCharts lies in building charts or graphs with simplistic visual regardless of device resolution as well as in providing a fallback compatibility for all current web-browsers.

Last but not least, socket.io, on the other hand is an upgraded version of WebSocket which provides compatibility across devices. With high performance and stability, it is obviously a flawless choice for real-time web application. Other than that, socket.io was also built on JavaScript foundation which makes a seamless combination with the other technologies used for this concept.

4 IMPLEMENTATION

In order to get the system operated, all the puzzle pieces have to be put together in a logical way, starting with installations of the infrastructure core. The following section will cover the initial steps to setup a basic skeleton for the app.

4.1 NodeJS and modules

NodeJS is available to almost every platform available including Windows, MacOS, Linux. The walk-through instruction is clearly available on [NodeJS.org](https://nodejs.org). However, for demonstration purposes, all the steps below this point will be taken part on MacOS. The visual instruction from the '.pkg' file downloaded to install NodeJS framework and Node package manager (NPM) altogether.

NPM is an essential part yet powerful of NodeJS environment. It allows us to customize the server to suit every needs from project to project. The main function of npm is to add or remove packages which are exclusively developed for NodeJS.

Table 2. NPM installing syntax.

```
$ npm install <PACKAGE_NAME>
```

Express.js framework

Express.js, which introduced after Ruby project Sinatra, is an essential framework for everyone to become familiar with NodeJS. According to Jim R. Wilson, Express provides most the plumbing code that coders will otherwise end up writing themselves [10]. With only few lines of code, a fully functional backbone was set up and ready to go. The concept behind Express is that it is predefined following MVC (Model View Controller) architecture with the support template engines.

Express Generator was created for beginners to jump into the developing phase without going through deeply configuration steps with Express by default. A basic web server will be deployed right out of the box after with just one line of code.

Table 3. Create project with Express Generator.

```
$ express sensorApp
```

We just created a sensorApp in the current directory with a simple web server with an index page.

Sensortag module

Sensortag module is provided through NPM and is installed by the command below:

Table 4. Sensortag module install command.

```
$ npm install sensortag
```

An example code comes in the package enable the server receives temperature and humidity from the sensor right out of the box. By making modifications in order to adapt to this project, specific functions regarding light sensor and gyroscope are also added. There are not only value receiving functions are implemented but a number of calculations are also placed in the code to calculate the output on the fly before it leaves for presenting.

Socket.io

The mechanism of socket.io consists of two parts: the server and client side. The server side is distributed from NPM. On the other side, the client one is slightly different. First, the library has to be embedded into the webpage.

Table 5. Link to “socket.io” library in HTML.

```
<script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>
```

Then functionalities are implemented within the JavaScript section.

Table 6. Get “socket.io” functions in JavaScript.

```
var socket= io.connect('http://yourdomain.address');
```

Additional modules

In addition to the mentioned packages above, there are some other dependencies needed to help the application operating such as: serve-icon, morgan, body-parser, cookies-MongoDB, assert.

4.2 JavaScript Libraries

mLab

mLab was renamed from MongoDB in February 2016 [11]. mLab is an online database service using MongoDB. It provides a convenient way to interact with MongoDB which is also compatible with many existing technologies. Easy to setup and reliable are advantages when using mLab. A registered account is needed to start using mLab.

Cloud-based databases are very popular right now due to their portability and convenience. mLab is a great example. For education or testing purposes, setting a standalone database-server could be time-consuming. mLab is considered as a better choice because of its simplicity in installation and reliability.

To set up, mLab created an URL to allow developers to access the database remotely. MongoClient is an object providing connecting functions to MongoDB database. By using it, manipulating the databases can be done with only few steps.

Table 7. Link to your mLab database.

```
url = "MongoDB://<adminuser>:<password>@<link to your
database:<port>/<database name>";
var MongoClient = require('mongodb').MongoClient;
MongoClient.connect(url, function(err, database)
{...});
```

HighCharts / HighStocks

There are plenty of JavaScript libraries for designing charts from our database.

HighCharts is one among the best so far when it comes to reliability and compatibility.

Table 8. Link to HighCharts Library in HTML.

```
<script src="https://code.highcharts.com"></script>
```

4.3 Wire up

With a foundation of the Express predefined server, the next stage is connecting all the components together.

By default, Express has its starting point set to 'www' file. Any activity followed by will derive from there. 'www' file's job is to configure a port for the server and handle related errors or exception.

App.js will act as a station where data flow will be processed and transferred to every part of the system. The purpose of app.js is to:

- Create and configure web server.
- Import libraries / plugins.
- Handle views.
- Connect with database.
- Process and manipulate data.

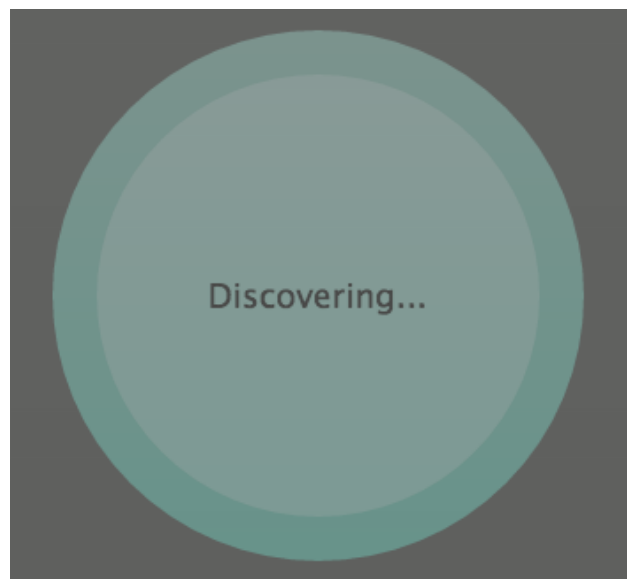
4.4 Setting up connection

In order to start the app, we issue this command in the app directory:

Table 9. Start NodeJS server command.

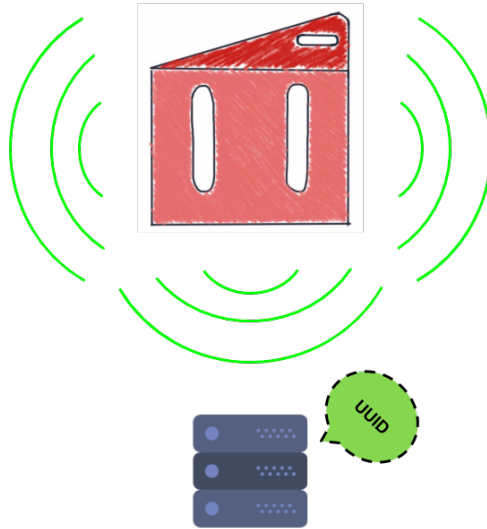
```
$ node app.js
```

Since the beginning, whenever the app is opened, a socket is set up to receive signals from the sensor every second which is called discovery mode. From the dashboard, the sensor status is presented as 'Discovering..' which means the connection between the sensor and server is not established yet.



Picture 4. Discovery mode.

The sensor will be turned on by pressing power button 3 for seconds. A blinking led light on the sensor indicates that it is in discovery mode. While in discovery mode, it starts broadcasting its address (UUID) to nearby devices.



Picture 5. Broadcasting connectivity signal.

With the help of sensortag library, the sensor was detected with the discovery function. From the console window, the sensor's type and device's ID were printed out to indicate that they found each other.

Device: cc2650

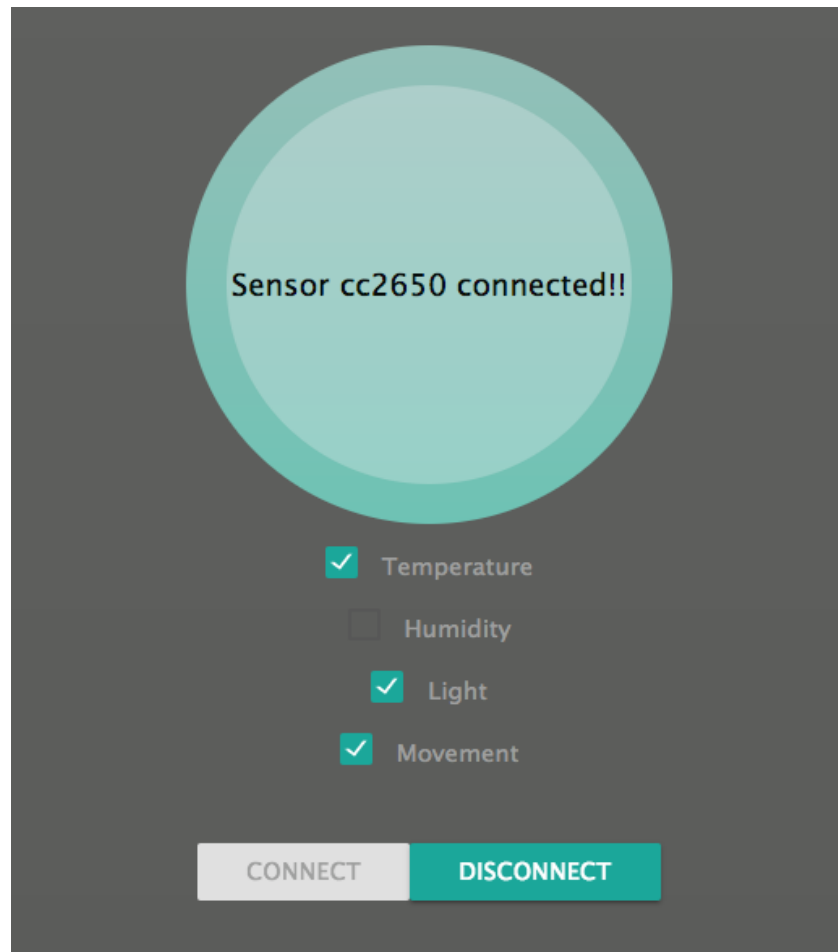
Device ID: ae68d4c95a9a41c7a197b040e1a7d757

left button PRESSED!

Picture 6. Sensor information recognized by node server.

In addition, another step was added to verify whether or not we are choosing the right sensor to connect to. After getting discovered, every time we press any button on the sensor, the server will show its recognition.

There are two methods to become established with the server. We can take a physical action directly on sensor by holding both buttons at the same time or, on the other hand, by clicking on connect button from the dashboard. The indication will be reflected on the server console. The user interface (UI) offers options to enable from only one to all the built-in sensors in order to adapt every need.



Picture 7. Sensor panel in dashboard.

Once the connection is up, temperature, humidity and light density sensors are activated accordingly. Measurements start to flood into node-server with an interval of 1 second. At the same time, MongoDB is also receiving those values then storing them within the collection named after the current date.

```

Sensor cc2650 is connected!
Device's ID detected: ae68d4c95a9a41c7a197b040e1a7d757
-----
Services have started !...
Humidity sensor Enabled !...
Temperature sensor Enabled !...
Luxometer sensor Enabled!...
-----
-----

          Temperature = 26.8 deg. C
          Humidity = 53.5 %H
          Object Temp = 6.3 deg. C
          Ambient Temp = 26.8 deg. C
lux value = 88.16
Adding value to database...

```

Picture 8. Responses from server when connected to SensorTag.

All of those flows of data are handled by socket.io. By setting up the bridges between the node server, the sensortag module and the UI, the numbers were transferred back and forth seamlessly between those ends which creates a data flow throughout the entire app.

4.5 Data Flow

Between UI and server

After users have made their choices of which type of data they want to retrieve from the sensor, a socket named “custom” was created in order to send that command straight to sensortag module. The module then enables the following sensors correspondingly.

```

io.on('connection',function(socket)
{
  socket.removeAllListeners();

  socket.on('custom',function(data)
  {
    sensortag(Number(data.status),data.temp0n,data.humi0n,data.lux0n,data.gyro0n);
    sStatus = data.status;
    console.log("Sensor status: "+ data.status);
    console.log("Temp on: "+ data.temp0n);
    console.log("Humi on: "+ data.humi0n);
    console.log("Lux on: "+ data.lux0n);
    console.log("Gyro on: "+ data.gyro0n);

    sigTemp = data.temp0n;
    sigHumi = data.humi0n;
    sigGyro = data.gyro0n;
    sigLux = data.lux0n;
  });
}

```

Figure 1. Data transfer from UI to server via “custom” socket.

In addition, another socket was initialized by the name of “valuesOut”. This socket is in charge of forwarding values which are received from sensortag module to the UI for presentation. The values are presented in gauges (meters) and in graphs.

```

socket.emit('valuesOut',{
  temp: sensortag.temp,
  humi: sensortag.humi,
  lux: sensortag.lux,
  gyro: sensortag.gyro,
  state: sensortag.state
});

```

Figure 2. Socket “valuesOut” structure.

Lastly, in order to keep track of how many sensors are operating, the “signal” socket is created. The role of this socket is to synchronize all the states of sensor to all the connected clients.

```

socket.emit('signal',{
  sta: sStatus,
  type: sensortag.type,
  sigTemp: sigTemp,
  sigHumi: sigHumi,
  sigLux: sigLux,
  sigGyro: sigGyro
});

```

Figure 3. Socket “signal”.

Between database server and node server

MongoDB, on the other hand, captures values right after they have arrived at the node-server. Along with the timestamp, each entry was input to the database server every second in a designated structure which is explained in later section.

```

var insertDocumentExplicit = function(db, callback) {
  getDayTime(gDate);
  if(sensortag.temp==null && sensortag.humi==null && sensortag.lux==null)
  {
  }
  else
  {
    db.collection("s"+gDate.d).insert
    ({
      "time": new Date().getTime(),
      "temp": Number(sensortag.temp),
      "humi": Number(sensortag.humi),
      "lux": Number(sensortag.lux),
      "state": Number(sensortag.state)
    }, function(err, result)
    {
      if(err){
        console.log("Error: "+err);
      }
    });
  }
};

```

Figure 4. Records are sent to database.

4.6 User interface

Dashboard

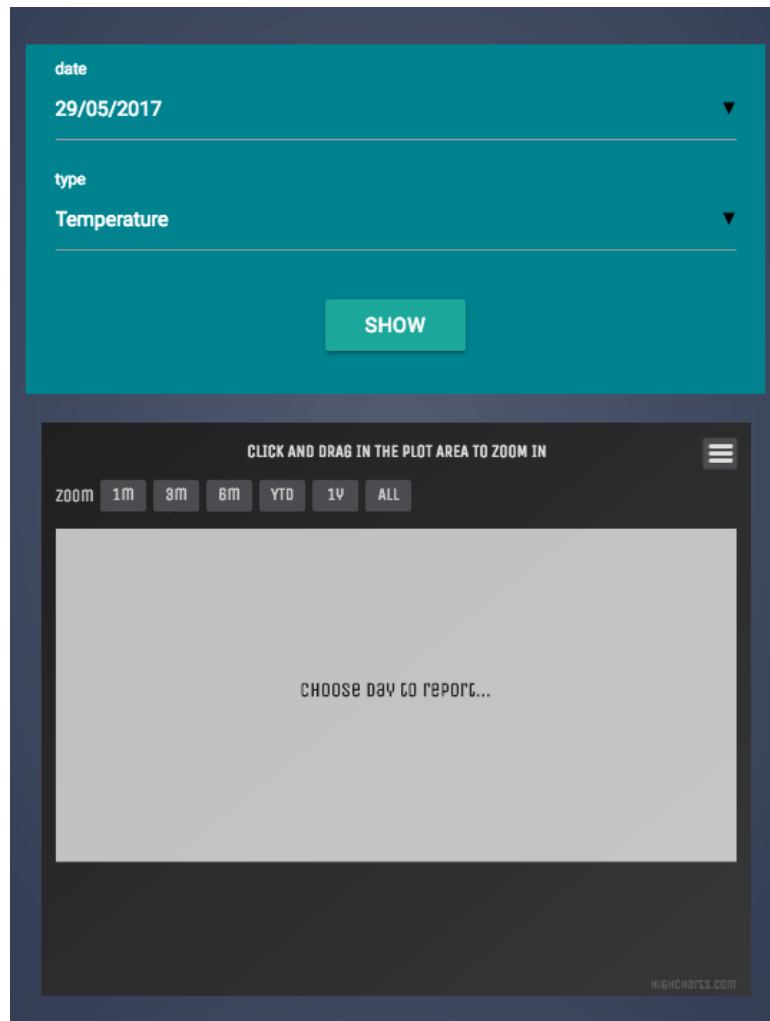
In terms of visualization of the current status, from the dashboard, all the values are updated every second in real-time. The chart in the section below is plotted in order to visualize every change since the sensor is connected.



Picture 9. Dashboard interface.

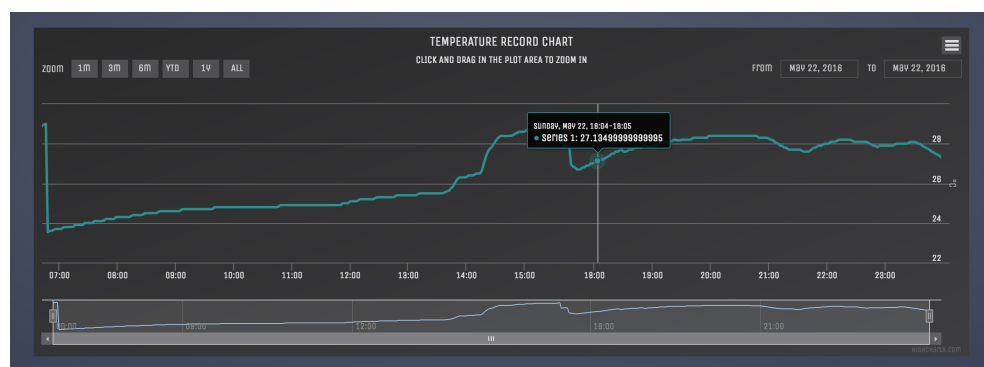
Report

This page was intended to present the data which is stored in the database server during the time being. We will have two fields to select: date and type of data. Since the structure of database in this project was categorized by date, users can choose which type of data they want to query on that specific day from the form above the graph.



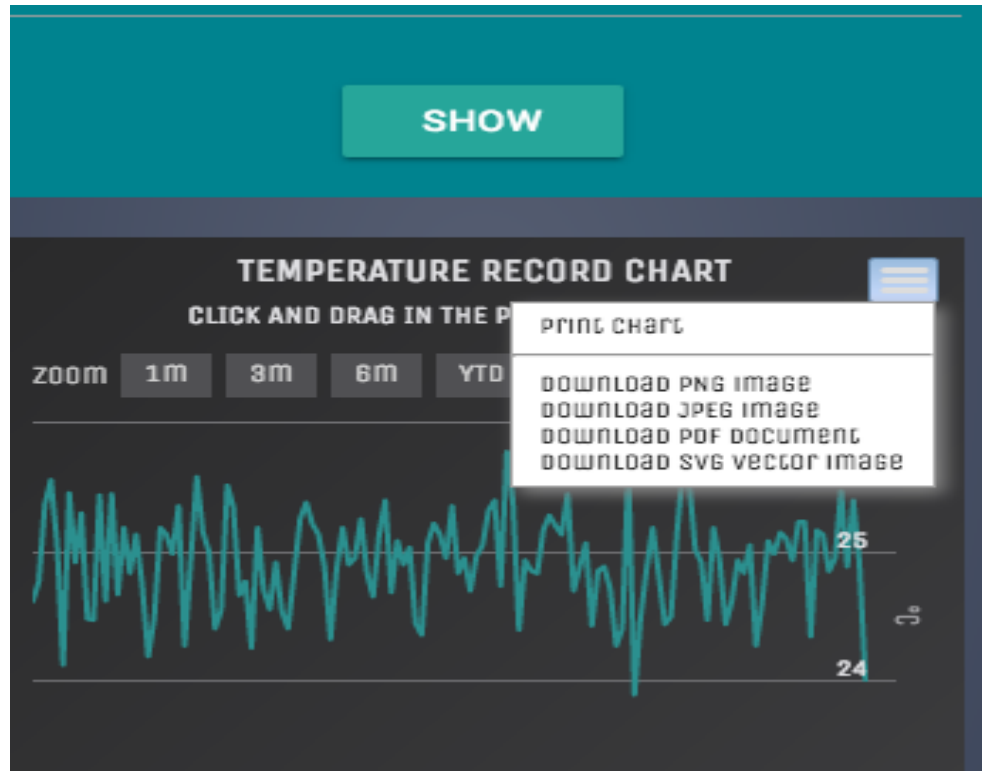
Picture 10. Show records panel.

After the "SHOW" button was pressed, requests are then forwarded to the MongoDB containing the type of data along with the date by user's choice to extract the records. After that, all the data will be converted into graph-friendly data format then is visualized by a graph as below.



Picture 11. Visualized graph of data.

One more integrated feature of HighCharts library is that we can export any graph to variety of image formats such as PNG, JPEG and PDF. These exported images can be used for storing or sending by email.



Picture 12. Exporting feature of HighCharts.

4.7 Database

When it comes to the application, the database plays an essential role. For this particular project, the database server is not only in charge of storing measurements from the sensor but also handling requests from user. Compared with the relational database, MongoDB has a different approach to build its own structure. As mentioned earlier, instead of tables, collections are used to store entries which are called keys. Each key consists of an object ID, time, measurements and each collection contains records for one day.

Database Name: **sensorApp**

Collection names: **name by date**

Keys:

- **Hour** - Integer
- **Minute** - Integer
- **Second** - Integer
- **Temp** - String

- **Humi** - String
- **Lux** - String
- **Gyros** - String

An example of a record in database is demonstrated in Picture 13 below.

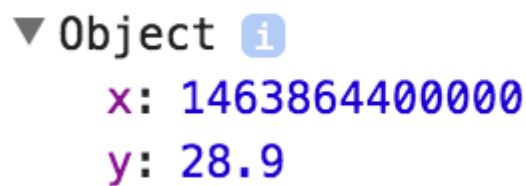
```
{
  "_id" : ObjectId("5740cc62a29ed90a8bcd6fc6"),
  "hour" : 0,
  "minute" : 0,
  "second" : 18,
  "temp" : "28.9",
  "humi" : "45.8",
  "lux" : "9.3"
}
```

Picture 13. Structure of a key in MongoDB database.

5 PERFORMANCE

Performance plays an important role in web applications and sensorApp is not an exception. A characteristic of this system is constantly retrieving values from the sensor with a tiny interval so a burdensome load of data will be carried by the database. Because of that, transferring such a huge amount of data to plot on the graph becomes challenging for both the server and client-side. Since every time users request a record for a long period of time, the database server has to make a query with a result can be up to thousands of entries or even more. Regarding the mechanism behind it, the database server processes the request by users from the dashboard, then returns with an array of objects accordingly. Before the array is sent back to the client-side, NodeJS converts the values to chart-compatible type so that the chart can present those later on.

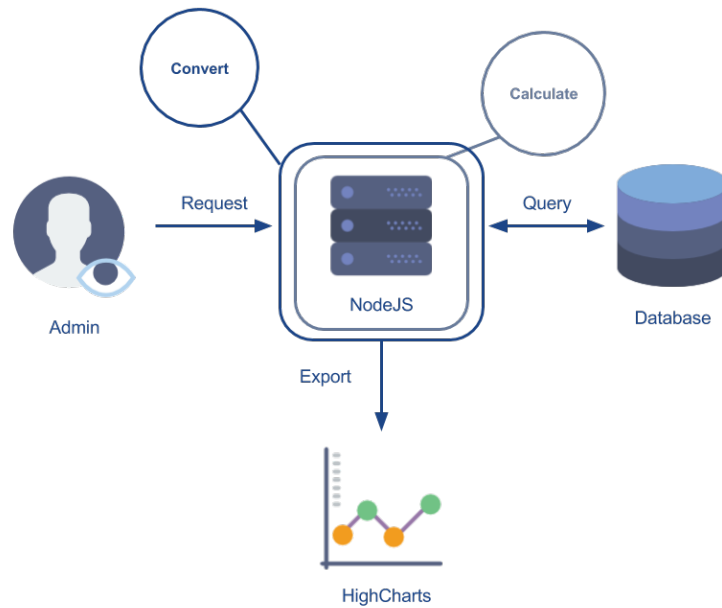
A point in the graph was technically represented by a time and a requested value which is either temperature, humidity or light density (lux). As presented in an example below, x is the timestamp (milliseconds) and y is the temperature (°C).

A screenshot of a data point object in a charting library. The text is displayed in a monospace font. It starts with a downward-pointing triangle followed by the word 'Object' and a small blue circle containing a white lowercase 'i'. Below this, the variable 'x' is followed by the value '1463864400000' in a purple color. Below that, the variable 'y' is followed by the value '28.9' in a purple color.

```
▼ Object ⓘ  
  x: 1463864400000  
  y: 28.9
```

Picture 14. A point transferred from server to graphs.

During the exporting data stage from database to client, an obstacle, which is a method of traveling values across the system, has been addressed. Since the amount of data is enormous, the processing speed will depend on many aspects namely querying, traversing, converting and plotting time. After trying different approaches, the final solution has performed well..



Picture 15. How data is handled upon requests.

The idea is that all the calculation as well as converting jobs will be given to the server which then transfers the result to the graph without any additional adjustment. After many attempts and modifications, the most efficient way to transfer data from the server to the UI is by using Ajax. Ajax call allows data traverse from one end to another without refreshing the web page. Therefore, user experience was improved dramatically. On the other hand, querying and inserting entries are handled simultaneously on the database server.

6 ASSESSMENT

The project covers many aspects of how to implement a real-time web application from scratch. With a combination of many recent technologies on the market, all the components cooperate smoothly. Moreover, with a simple yet practical user interface, the monitoring task now has become more transparent. In fact, the requirements enable the author to learn more and have a chance to apply programming skills as well as achieve a significant amount of knowledge throughout the making. Moreover, troubleshooting or problem solving skill was gained gradually while facing obstacles on the way.

Two months including researching time is the total amount time to build everything from an idea to the fully working app. It took 1 week at least during the timeline to test out all the components as well as combine them together. The final testing and tuning up phases took another week in order to ensure the outcome met the expectations.

6.1 Obstacles

Although the app has been functional, there are still problems that need to be addressed.

First of all, the measurement completely depends on the sensor. During the debugging stage, the provided sensor has displayed a strange behavior. For example, the humidity sensor randomly sends its peak values which are -40 degree Celsius and 100% humidity to be precise to the server instead of its current values. Despite the newest firmware installed or fresh battery installation, no improvement has been seen. Overall, it most likely could be a hardware fault.

Secondly, the performance of database has its own downside. The problem could be avoided by a high speed connection between the client and the server while querying history records from database. Because of a huge amount of entries flooding into the same socket, it can cause latency in response.

Lastly, since the server does not have any encryption method to bond with, security matters should be taken into account. No credential is required to access the dashboard remotely so it is totally possible if outsiders want to interfere with the operation of the system.

6.2 Potential Future Improvement

Everything has its positive and negative side and, of course, this app is not an exception. A considered number of tasks have been accomplished

flawlessly. However, there were some improvements that could have been made to take it to another level.

Firstly, technically, an accuracy of light detection needs to be increased. At the moment, the concept of tracking opening/closing door movements intuitively depends on the fluctuations of the line chart. Theoretically, the lighting level is completely different at any random point in the medium. Because of that, every time the door was opened/closed, the changes were reflected by the upward or downward trend on the graph.



Picture 16. Light density presentation.

Secondly, an administrator may need an analytical report for a longer period not just a day. Because of that demand, the exporting records feature should be upgraded to a larger scale which allows the administrator to review a statistic for a week, a month or even a year. Implementation does not seem to be sophisticated after all. However, the transmitting speed of records across the network should be taken into account. For a full day, we have 86 400 records with 1s interval to be exact. On the other hand, roughly, $2.592e+6$ records are registered in a month or even a excessive numbers of records in a year. In addition, the struggle is not only with the bandwidth of network but the numbers of connections accessing the app in order to export at the same time is also a challenging problem. The performance could be improved by using efficient tweaking methods.

6.3 Limitations

Power consumption

The sensor is required to function constantly to keep track of every event in real time. As mentioned, the limit in battery capacity is a serious concern. Throughout the testing phrase, with 1s interval for transmission

and the use of one coin-size battery, the sensor lasts only a few days before the battery runs out. That result was conducted when all the sensors were enabled, namely temperature, humidity, luxometer and gyroscope. The more sensors are enabled, the more power the app consumes.

According to Misha and her analysis [12], the power consumption of the sensor throughout operation stages was reflected by the table below.

Table 10. Power consumption.

	Current in milliamp (<i>mA</i>)
Powering on	12
Standby	0.24
Temperature sensor ON	0.84
Temperature + Humidity sensors ON	0.92
Light sensor (Luxometer)	0.56
Motion sensor	4.16
Barometric sensor	0.5
All sensors on (100ms sample rate)	~5.5

Following the analysis, the 240mAh battery can only power the sensor for less than 48 hours with maximum transfer rate or roughly 240 hours with just only temperature and humidity sensors on. This result is far from what Texas Instruments promotes on their website which it can last for a year. Therefore, in order to accomplish that rate of maintenance, we can use an external power supply. Because of lack of physical interface, the sensor needs support from the development kit to connect to external battery (power bank) via USB port.

Gyroscope accuracy

Because of the sensitivity of light sensor (luxometer), every minor movement of the door will be registered. Therefore, depending on light density value alone to determine door movement is not practical in a real life scenario. However, combining that data with values from gyroscope makes the determination much more accurate. For example, when the door starts to move, the gyroscope reflects 3 values from 3-axis and, simultaneously, luxometer provides fluctuations from light density. SensorTag comes with different sensors dedicated to motion detection such as accelerometer and gyroscope. In order to track the object's movements, in this case is the door, the gyroscope sensor was enabled. This sensor has 3 axis variables similar to accelerometer which are x, y and z. Depending on the position the SensorTag was placed on to the door, one of the 3 variables would change accordingly to the door movements. By calculating the difference between 2 values when the door at 2 positions, we can determine whether or not the door is being opened.



Picture 17. Gyroscope presentation.

Regarding the calculation method, the values of 3 axis (X_0 , Y_0 , Z_0) when the door at rest were initially measured then subtracted with values (X , Y , Z) when the door started to move. The differences between the initial values and current values of 3 axis are called Delta1, Delta2, Delta3.

$$\Delta 1 = | X_0 - X |$$

$$\Delta 2 = | Y_0 - Y |$$

$$\Delta 3 = | Z_0 - Z |$$

Equation 1. Differences when sensor changing position.

Since the sensor is extremely sensitive, even a small movement can make a difference. Therefore, a so-called Offset Constant (C) is also created to exclude those tiny chances from algorithm. In this case 0.8 is the offset constant. With that, we can decide whether the door is opened or closed by comparing Deltas to Offset Constant (C).

Table 11. Sensor states decided by Delta.

	Delta1	Delta2	Delta3
> C	<i>Moving</i>	<i>Moving</i>	<i>Moving</i>
<= C	<i>At rest</i>	<i>At rest</i>	<i>At rest</i>

7 CONCLUSION

As the result, a fully functional monitoring server meets the initial objectives:

- Measures environmental variables namely temperature, humidity, light density, gyroscope values from the SensorTag.
- Has a real-time visualized graphical interface.
- High-capacity database provides analytical data in form of graphs with exporting features built-in.
- Provides a managing interface can be remotely accessed widely from anytime, anywhere.
- Is fully compatible with most of browsers and devices.

At the time this thesis was written, this approach to SensorTag devices is not popular. Most of recent projects use mobile device as a relay device to forward information from sensor to the “cloud”. Users then have to use web service from the provider in which it is implemented to review the received data. Most services do not provide the database extraction feature.

With the help of a combination of NodeJS server, MongoDB, socket.io and Front-end frameworks, we can almost establish all the web application infrastructure not only in a commercial but also in an industrial environment. This app is a definite example of the Internet of Things in practice which is a trend in embedded programming nowadays.

In conclusion, the application is using all the modern technologies so it is practical in production. It runs properly and meets all the requirements as expected even though there are still limits in technology as well as the experience of the author. In order to complete the stack, many skills have been required including fluency in programming languages, manipulating and handling big data in a network. Moreover, communication between physical device and the API is also a challenge.

All in all, there is spacious room for improvements in the future. Despite the size of this project, the versatility and possibility are endless if we can keep the foundation and implement specific infrastructure on top to suit every need.

REFERENCES

- [1] Texas Instruments Sensortag. Consulted 30.3.2016. Available at <http://www.ti.com/tool/cc2650stk#1>.
- [2] SimpleLink SensorTag Debugger DevPack. Consulted 30.3.2016. Available at <http://www.ti.com/tool/cc-devpack-debug>.
- [3] Bluetooth BLE. Consulted 30.3.2016. Available at <https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works>.
- [4] Mithun Satheesh, Bruno Joseph D’mello, Jason Krol. 2015. Web Development with MongoDB and NodeJS. 2nd Edition. Packt Publishing. E-book. Consulted 5.4.2016.
- [5] Sandeep Mistry. 2015. Node-sensortag. Consulted 15.4.2016. Available at <https://github.com/sandeepmistry/node-sensortag>.
- [6] Fionn Kelleher. 11.8.2014. Consulted 20.4.2016. Available at <https://nodesource.com/blog/understanding-socketio/>.
- [7] Ryan Benedetti, Ronan Cranley. 2011. Head First jQuery. 1st Edition. O’Reilly Media, Inc. Consulted 20.4.2016
- [8] MongoDB. <https://docs.MongoDB.com/manual/>. Consulted 25.4.2016
- [9] Joe Kuan. 2015. Learning HighCharts 4. Packt Publishing. E-book. Consulted 5.5.2016.
- [10] Jim R. Wilson. 2013. Node.js the Right Way. The Pragmatic Programmers. Consulted 10.5.2016.
- [11] mLab. Consulted 11.5.2016. Available at <https://en.wikipedia.org/wiki/MLab>.
- [12] Misha, 9/6/2015. TI SensorTag 2 Power consumption analysis. Consulted 30.5.2016. Available at <http://mobilemodding.info/2015/06/ti-sensortag-2-power-consumption-analysys/>.

Appendix 1 - Application tree

```
.
├── app.js
├── package.json
├── bin
├── ┬── www
├── public
├── │   ├── images
├── │   ├── javascripts
├── │   │   ├── dashboard.js
├── │   │   ├── socket.io.js
├── │   │   ├── sensortag.js
├── │   │   └── report.js
├── │   └── stylesheets
├── │       ├── style.css
├── │       └── report.css
├── routes
├── views
├── │   ├── error.pug
├── │   ├── dashboard.pug
├── │   ├── report.pug
├── │   └── layout.pug
```

Appendix 2 – Sensortag module code

```

var async = require('async');
var SensorTag = require('sensortag');// sensortag library
var status=0;
var tOn,hOn,lOn,gOn;
SensorTag.discover(function(tag) {
    // when you disconnect from a tag, exit the program.
    function disconnectTag()
    {
        tag.on('disconnect', function() {
            console.log('disconnected!');
            module.exports.dis = 1;
            process.exit(0);
        });
    }
    function connectAndSetUpMe() {
    // attempt to connect to the tag
    console.log('connectAndSetUp');
    tag.connectAndSetUp(enableIrTempMe);
    // when you connect, call enableIrTempMe
    }
}

function enableIrTempMe(tempON,humiON,luxON,gyroON)
{
    if(tempON)
    {
        console.log("Temperature sensor is enabled!");
        tag.enableIrTemperature();
    }
    if(humiON)
    {
        console.log("Humidity sensor is enabled!");
        tag.enableHumidity();
    }
    if(luxON)
    {
        console.log("Luxometer sensor is enabled!");
        tag.enableLuxometer();
    }
    if(gyroON)
    {
        console.log("Gyroscope sensor is enabled!");
        tag.enableGyroscope();
    }
}

```

```

}
function disableSensors()
{
    // disable all sensors
    tag.disableHumidity(tag.disableLuxometer(tag.disableIrTemperature(tag.disableGyroscope())));
    console.log("Disable all sensors!!!");
    disconnectTag();
}
function notifyMe(tempON,humiON,luxON,gyroON) {
    console.log('Sensor '+tag.type+' is connected!');
    console.log('Device\'s ID detected: '+tag.id);
    console.log('-----');
    console.log("");
    tag.unnotifySimpleKey();
    if(tempON)
    {

        tag.notifyIrTemperature(tag.setIrTemperaturePeriod(1000,listenForTemp
Reading));
    }
    if(humiON)
    {
        tag.notifyHumidity(tag.setHumidityPeriod(1000,listenForHumidity));
    }
    if(luxON)
    {

        tag.notifyLuxometer(tag.setLuxometerPeriod(1000,listenForLuxometer));
    }
    if(gyroON)
    {
        tag.notifyGyroscope(tag.setGyroscopePeriod(1000,listenForGyroscope));
    }
}
function listenForLuxometer(){
    // Listen for Luxometer
    tag.on('luxometerChange', function(lux)
    {
        // console.log('lux value = ',lux);
        module.exports.lux = lux.toFixed(1);
    });
}
function listenForGyroscope(){
    // Listen for Luxometer
    var tempX,tempY,tempZ,state;

```

```

tag.on('gyroscopeChange', function(x,y,z){
    // console.log('Gyroscope:');
    // console.log('x: ',x.toFixed(1));
    // console.log('y: ',y.toFixed(1));
    // console.log('z: ',z.toFixed(1));

    module.exports.gyro = x.toFixed(1) + " | " + y.toFixed(1) + " | " +
z.toFixed(1);
    if(x.toFixed(1)-tempX>0.8)
    {
        state = 1;    }
    else
    {
        state = 0;    }
    module.exports.state = state;
    tempX = x.toFixed(1);
    tempY = y.toFixed(1);
    tempZ = z.toFixed(1);
    });
}
// When you get an accelermeter change, print it out:
function listenForTempReading() {
    tag.on('irTemperatureChange', function(objectTemp, ambientTemp) {
        // console.log('\tObject Temp = %d deg. C', objectTemp.toFixed(1));
        // console.log('\tAmbient Temp = %d deg. C',
ambientTemp.toFixed(1));
        var intemp = ambientTemp.toFixed(1);
        module.exports.temp= ambientTemp.toFixed(1);
    });
}

// Get data from Humidity Sensor ( + Temperature )
function listenForHumidity() {
    tag.on('humidityChange', function(temperature, humidity) {
        // console.log('\tTemperature = %d deg. C', temperature.toFixed(1));
        // console.log('\tHumidity = %d %H', humidity.toFixed(1));
        module.exports.humi= humidity.toFixed(1);
        var intemp = temperature.toFixed(1);
        var inhumid = humidity.toFixed(1);
    });
}
// when buttons pressed
function listenForButton() {
    tag.on('simpleKeyChange', function(left, right) {
        console.log("Device: "+tag.type);
        console.log("Device ID: "+tag.id);
    });
}

```



```

        if (left) {
            console.log('left button PRESSED!');
        }
        if (right) {
            console.log('right button PRESSED!');
        }
        // if both buttons are pressed, disconnect:
        if (left && right) {
            console.log("Device: "+tag.type+" with id of:
"+tag.id+" connected !");
            enableIrTempMe();
            // tag.disconnect();
        }
    });
}
// Now that you've defined all the functions, start the process:
tag.connectAndSetUp(
    function(){
        var intervalID=setInterval(function()
        {
            // console.log("Check status: ",status);
            if(status==1)
            {
                notifyMe(tOn,hOn,lOn,gOn);

                enableIrTempMe(tOn,hOn,lOn,gOn);
                // connected signal

                clearInterval(intervalID);
                var secondIntervalID =

setInterval(function(){
                    if(status == 2)
                    {
                        disableSensors();

                clearInterval(secondIntervalID);
                    }
                },1000);
            }
        },1000);
        console.log("Sensor Type: ",tag.type);
        console.log("Sensor ID: ",tag.id);
        module.exports.sta = status;
        module.exports.type= tag.type;
        tag.notifySimpleKey(listenForButton); // start the button listener);
    }
);

```

```
    });  
  });  
  module.exports= function(s,tempOn,humiOn,luxOn,gyroOn)  
  {  
    status=s;  
    tOn = tempOn;  
    hOn = humiOn;  
    lOn = luxOn;  
    gOn = gyroOn;  
  }  
}
```