jamk.fi

Remote monitoring of embedded device

Tomáš Urban

Bachelor's thesis
May 2018
School of Technology, communication and transport
Degree Programme in Information and Communications Technology
Software Engineering

jamk.fi

Description

Author(s)	Type of publication	Date
Urban, Tomáš	Bachelor's thesis	May 2018
		Language of publication: English
	Number of pages 46	Permission for web publication: yes

Title of publication

Remote monitoring of embedded device

Degree programme

Information and Communications Technology, Software Engineering

Supervisor(s)

Lappalainen-Kajan, Tarja

Assigned by

Mika Rantonen

Abstract

This paper introduces the process of the design and implementation of an application for monitoring an embedded device. The project consists of three main applications and usage of IoT platform for visualizing the results from the device. The firmware for the embedded device is written in C programming language, and the applications for Raspberry Pi are bound together. The server-side application is written in NodeJS is RESTful and can be connected with any other application.

The first part of the thesis interpretes the theoretical part of the project, the concept of embedded devices and their different usage. The architecture of RESTful API is introduced and different kind of sensors are explained.

The firmware for the embedded device was developed first, and it was followed by Python application and server-side application. The embedded device and Raspberry communicate via Bluetooth technology. Communication with the server and with ThingsBoard is implemented with HTTP and MQTT protocols. For data storage, NoSQL database is used. ThingsBoard provides a user-friendly interface for visualizing data from the device.

With the device a fully working application was developed where data is successfully recorded, stored and serves the user in readable and useful way.

Keywords/tags

REST, Accelerometer, Magnetometer, NodeJS, Compass, ThingsBoard, Embedded system

Miscellaneous

Content

Figures

Acronyms

AC/DC	Analog/Digital converter
CoAP	Constrained Application Protocol
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
DAC	Digital/analog converter
HATEOAS	Hypertext As The Engine Of Application State
IC	Integrated Circuit
JSON	JavaScript Object Notation
MEMS	Micro-Electro-Mechanical Systems
MISO	Master in slave out
MOSI	Master out slave in
UI	User Interface
UART	Universal asynchronous receiver transmitter
RAM	Random-access memory
REST	Representational State Transfer
RISC	Reduced Instruction set Computer
RTC	Real-time clock
SCL	Serial clock
SDA	Serial data
SS	Slave select

1 Introduction

This thesis deals with a full-stack project for monitoring an embedded device. This thesis investigates how the process of measurement, forwarding and visualizing data can be implemented. The selected features of microcontrollers are discussed on a theoretical level. The implementation of peripherals such as USART I2C into the embedded device is described in following chapters. The characteristics and constraint of a REST API are discussed, and the main features are implemented into the project.

When talking about the embedded device, the developed device can be considered as IoT device. IoT devices are typically self-standing devices with connectivity to the network. They are used for monitoring the environment control another device or measure some sort of variables. The general principle is to connect these devices to one central point from where a variety of data are processed and involve the desired action.

The main goal of this paper was to create a fully working project from backend applications to frontend application. The project consists of three main self-standing application. The application for an embedded system written in C programming language designed for 8-bit microcontroller ATMega328. Python script runs on Raspberry Pi and, a server-side application is written in NodeJS.

The third chapter discusses sensors. Attention was given to two kinds of sensors, namely accelerometer and magnetometer. Both were implemented into the embedded device and used in the project. Also, these kinds of sensors are very often used in many different application and handle reading data as the crucial part of the correct implementation.

The fourth chapter discusses essential features of a REST API. The process of developing a quality API with necessary components is introduced and explained.

The fifth chapter contains all theoretical information from the previous chapter put together, and the building of an entire fully working project.

2 Microcontrollers

2.1 Microcontrollers in general

The microcontroller is a computer. When talking about computers, whether about a personal computer, microcontroller or a large enterprise server, they all have several features in common: (Brain 2000)

- Every computer has a Central Processing Unit (CPU) that executes programs. Self-standing application running or particular operation system, both are programs executed by the CPU.
- The CPU has to load a program from memory where the program is saved. In a personal computer, it is the hard drive, and in a microcontroller, it is the flash memory.
- Every computer needs to communicate with its environment. Input and output devices represent a gate to outside. Printer and monitor are output devices, keyboard and mouse input devices. There are also devices which handle both input and output, e.g. hard drive.

Microcontrollers include all necessary components including a CPU. One package contains memory, such as flash memory, Random-access memory (RAM), peripherals as serial communication ports, input-output ports, converters. Microcontrollers regardless of their quite low computer power are often implemented into systems for complex problems. Microcontrollers are usually applied in home devices, vehicles, IoT devices, wearable electronics and medical devices. The word system-on-a-chip is often associated with the higher end of microcontrollers, despite of no precise parameters of memory or clock speed. (Rouse 2006) Microcontrollers can be found inside a surprising number of products. Usually, they are a part of a more complex system called embedded system.

2.2 History

The first single-chip microprocessor was the 4-bit Intel 4004 released in 1971 followed by other more capable processors over the years. These, however, all required external chips to implement a working system, raising the total system cost, and made it impossible to economically computerize appliances. (Wikipedia Microcontrollers)

The first microcontroller was introduced in 1974 by Texas Instruments. It was a family of 4-bit microcontrollers called TMS1000. All microcontrollers had a very simple design and implemented just 2-4-bit registers with a limited number of instructions. They do not support interrupts. (Texas Instruments 2001)

Intel released an 8-bit microcontroller named as 8051 in 1981. It was referred to as a “system on chip” and it contains all major features implemented in microcontrollers. On single chip is implemented with (Texas Instruments 2001):

- A serial port
- 128B of RAM
- Two timers
- 4 input/output ports (each 1B)
- 4KB of on-chip ROM

Figure 1 illustrates the 8051 Microcontroller architecture.

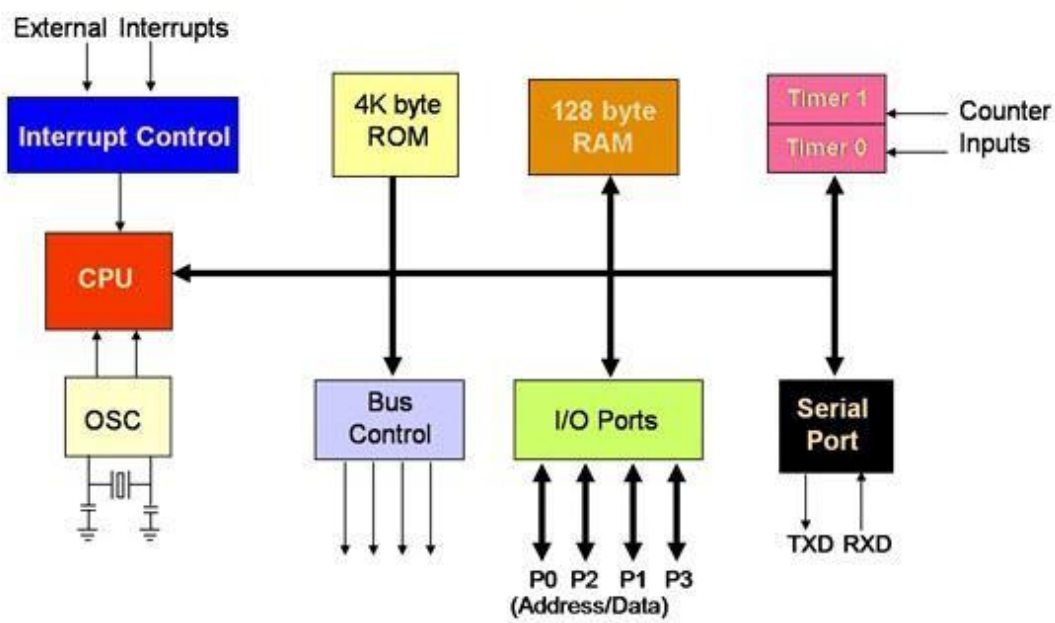

Figure 1 Block diagram of 8051 Microcontroller (The 8051 Block Diagram 2011)

2.3 Embedded system

An embedded system is a connection of two separate parts in one device. Computer hardware and software are developed especially for a particular device. Both are designed for one or more particular functions within a complex system. Small devices such as cameras, toys, household appliances, clothes, medical equipment or bigger automobiles and airplanes are all possible locations for an embedded system. In general, embedded systems are computing systems; however, they can have various purposes. From elementary devices designed to perform a single task, e.g. measure temperature, humidity or other physical variables to a complex device such as in tablets, mobiles with complex UI. A device usually includes buttons, touchscreen, LEDs and more. Some of the systems can use a remote user interface as well. (Rouse 2016)

Embedded systems can be separated into two groups according to their CPU. The categories are either microprocessor-based or microcontroller-based. In any of those

cases, the heart of the product is an internal circuit (IC), that is typically constructed to execute the computation for real-time operations. There is no visual difference between microprocessors and microcontrollers. The main difference depends on their internal implementation. Internal structure of the microprocessor implements nothing just a central processing unit and so the attachment of other necessarily components e.g. peripherals and memory chips are required. A microcontroller is designed to operate as self-contained systems. (Rouse 2016)

One of the first real modern embedded systems was Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumentation Laboratory in the early 1960s. The system was designed for guidance, navigation, and control of the spacecraft. The size of the device was at that time relevant and saved plenty of space and weight on board. The dimensions were 61x32x17 cm with the weight of 32kg. Since these first applications in the 1960s, price dimension and also processing power and functionality dramatically started to increase. (Wikipedia Embedded system)

2.4 Classification

Microcontrollers are characterized by their bus-width, instruction set, and memory structure. All these characteristics displayed below in Figure 2 describe the different types of microcontrollers. There may be some other classification, however, most of the types can be seen through this crucial classification. (EnTcians 2013)

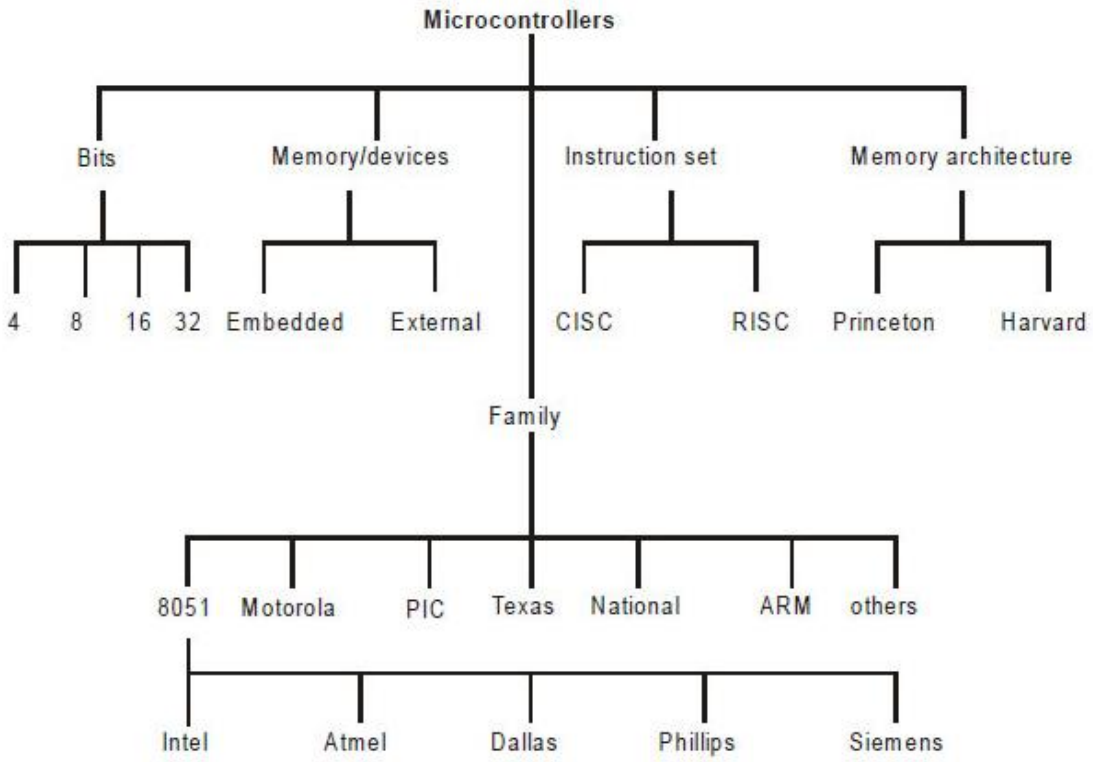

Figure 2 Types of Microcontrollers (EnTcians 2013)

Microcontrollers can be 8-bit, 16-bit, 32-bit or higher. The number of bits represents the length of the instruction which CPU if able to execute in one operation. 32-bit or higher microcontrollers use longer instructions to perform arithmetic and logic operations and generate a higher computer power. (EnTcians 2013)

When an embedded system including all the hardware and software sections in a single unit, the MCU is called embedded microcontroller. A crucial part of the embedded microcontroller is that no other or just very few external systems is present for processing. For example, a telephone handset circuit uses an embedded microcontroller. (EnTcians 2013)

Embedded system includes an MCU where not all hardware is included into one chip but all or part of the memory unit externally interfaced using an interfacing circuit called the glue circuit, the MCU is called an external memory microcontroller. For example, 8031 has the program memory interfaced externally to it. (Basheer 2013) Microcontrollers can also be divided according to their family. Family is a company which produces a particular type of microcontrollers. Some of the companies have

focused on specific types of microcontrollers, e.g. analog devices, specializing in data conversion and signal processing technology. The best-known companies are e.g. Atmel, Analog Devices, Intel, NXP Semiconductors, Panasonic, STMicroelectronics, Texas Instruments. (Basheer 2013)

Next important criterion is the instruction set. The instruction set defines possible uses of a microcontroller. Microcontrollers with Reduced Instruction set Computer (RISC) set of instruction are valuable in specific applications where there is no need for bigger variability. On the other hand, Complex Instruction set Computer (CISC) set of instruction offers all set instructions on the CPU, which makes microcontroller more versatile and eligible for many different tasks. (ARM Articles 2008)

One of the basic separations is memory architecture. The point when a microcontroller unit has a dissimilar memory address space for the program and data memory; the microcontroller has Harvard memory architecture in the processor. The second variant is the one where the microcontroller has one common memory address available for data memory and program memory called Princeton architecture. (ARM Articles 2008)

2.5 Peripherals

The microcontrollers additionally include circuitry that implements a variety of peripheral functions, enabling easier deployment in a variety of settings. Microcontrollers usually include a variety of input and output ports, e.g. to facilitate the signal flow between the microcontroller and external sensors and switches. It also usually includes one or more ADCs to convert incoming analog signals into digital values, and one or more DACs to convert digital values into output analog signals. These I/O ports and converters enable the use of a variety of signal types. Figure 3 shows microcontrollers conceptual configuration.

Another ubiquitous peripheral is the real-time clock (RTC), which is used to enable accurate time measurements and time-of-day monitoring, and it is widely utilized by

processes that refer to or are dependent on time. Still, another common peripheral is the UART, used to convert parallel signals into serial, and serial into parallel. (MCU programming 2017) Figure 3 illustrates the conceptual configuration of a microcontroller's peripherals and their connection to the main bus.

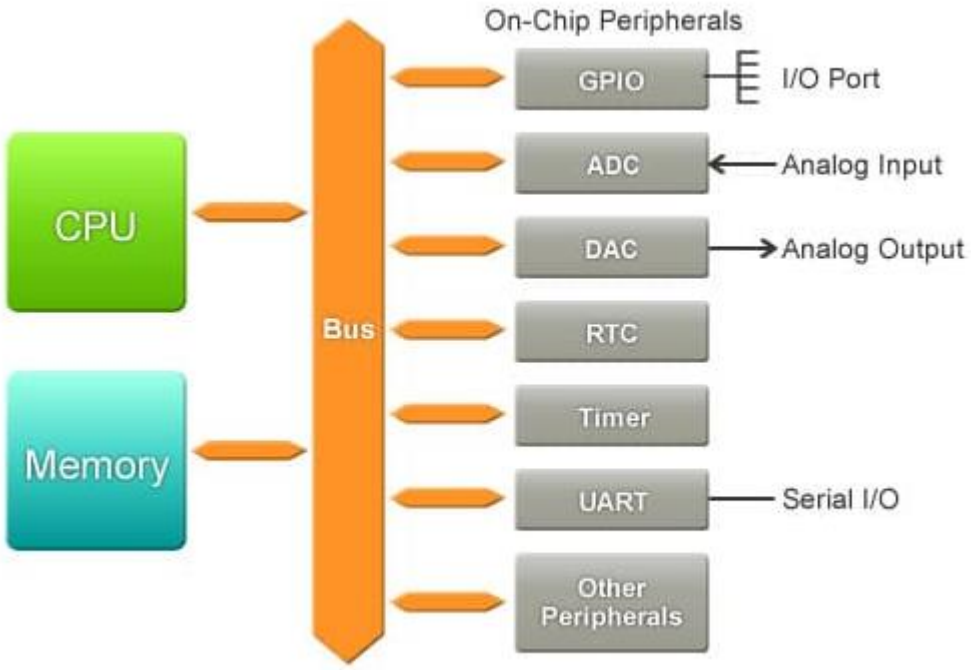

Figure 3 Microcontrollers Internal Conceptual Configuration (MCU Programming 2017)

2.5.1 USART

Universal Synchronous/Asynchronous Receiver and Transmitter is a communication peripheral able to work in two different modes depending on the configuration. The transmission of data is handled on TX pin and received on RX pin. USART mode can be set up to run in synchronous mode. In this mode, the sending peripheral will generate a clock that the receiving peripheral can recover from the data stream without knowing the baud rate ahead of time. Alternatively, the link will use a completely separate line to carry the clock signal. Standard data transmit Through UART is described below as follows: (Wikipedia USART)

- Start bit: First bit defined as a logic value 0. After this bit receiver should expect a first valuable bit of a message.
- Data bits: After start bits follow 5-8 data bits. LSB bit as first and MSB as last one. If no Parity bit is set, the data frame can be 9 bits long.
- Parity bit: is a way for the receiving UART to tell if any data has changed during transmission.
- To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for at least two-bit durations.

Although USART has many disadvantages such as limited length of the data frame to 9 bits, communication takes place only between transceiver and receiver without supporting multiple slaves, or the multiple master system is in many cases a fitting method for communication and still widely used. The advantage is the easy configuration since only two wires are necessary, and no clock signal is necessary. USART can implement parity bit for error checking, and the structure of data packet can be changed as long as both sides are set up for it.

2.5.2 SPI

"Serial Peripheral Interface is a synchronous serial communication interface developed for short distance communication." SPI is used usually inside one board when devices use full-duplex communication with master-slave architecture. Only one master can communicate with many slave devices. SPI bus is specified with four signals: (Wikipedia SPI)

- MOSI
- MISO
- SCKL
- SS

Signal SCLK is managed by the master device. The serial clock signal is transmitted from master to all slave devices. The clock rate is selected by master according to the

slowest slave device. Master in slave out (MISO) is the data output from the slave. Master out slave in (MOSI) is data output from the master. Slave select signal defines which slave device is active and can communicate with the master. The connection between two devices can be set up to logical value 0 and does not have an effect on communication. (Wikipedia SPI)

2.5.3 I2C

The I2C or two wire interface bus is a bi-directional interface used by microcontrollers. Microcontroller as a master device in connection can communicate with many slave devices through one bus. Data transmitted from slave device can not be send to the master device until slave has been addressed. I2C interface uses a specific 7-bit address for every device connected to the bus. Address is a uniqe for each device on the same I2C bus. “Most of the slave devices will require configuration upon startup to set the behavior of the device, typically done when the master accesses the slave's internal register maps, which have unique register addresses.” I2C bus requires only two wires. SDA wire is used for the master and slave to send and receive data. SCL wire that carries the clock signal. (Understanding the I2C 2015)

3 Mems Sensors

3.1 Overview

Micro-Electro-Mechanical Systems (MEMS) is a technology that in its most general form can be defined as miniaturized mechanical and electro-mechanical elements that are made using the techniques of microfabrication. The one main criterion of MEMS is that there are at least some elements having some sort of mechanical functionality whether or not these elements can move. While the functional elements of MEMS are miniaturized structures, sensors, actuators, and microelectronics; the most notable elements being the microsensors and microactuators. Microsensors and microactuators are appropriately categorized as “transducers”, which are defined as devices that convert energy from one form to another. In the case of microsensors, the device typically converts a measured mechanical signal into an electrical signal. Typical MEMS devices are (Hemingway 2015):

- Sensors
    - Pressure Sensors
    - Accelerometers
- Actuators
    - Gyroscopes
    - High Aspect Ratio Electrostatic Resonators
    - Thermal Actuators
    - Magnetic Actuators
    - Comb-drives (Hemingway 2015)

3.2 Accelerometer

An accelerometer is a compact sersor designed to measure non gravitational acceleration, or vibration. “The force caused by vibration or a change in motion (acceleration) causes the mass to "squeeze" the piezoelectric material, which

produces an electrical charge that is proportional to the force exerted upon it." Accelerometers are widely used in robotics (position measurement, angle), air force(autopilot), automotive industry (airbags, antilock brake system, ride control system), mobile electronics, game consoles. They are used together with a gyroscope for body movement monitoring. There are various different types of accelerometers, e.g. laser accelerometer, a piezoelectric accelerometer, and surface micro machined capacitive (MEMS). These days, a new group of accelerometer called the Micro-Electro-Mechanical System (MEMS) Accelerometer becoming more popular. Their biggest advantage is a simple architecture, reliability and low price. (Omega company 2003) "It consists of a cantilever beam along with a seismic mass, which deflects due to the applied acceleration. This deflection is measured using analog or digital techniques and will be a measure of the applied acceleration." (Accelerometer 2011)

Before selecting an accelerometer for a given application, it is important to consider some of its key characteristic (Accelerometer 2011):

- Frequency Response – is detected by analyzing the features of the quartz crystal included into sensor or the resonance frequency of the device.
- Accelerometer Grounding may be visible in two different foms. The first one is called the Case Grounded Accelerometer. This kind of device is sensitive to ground noise. The second form is ground Isolation Accelerometer. This device is isolated from the cover. This device is a very susceptible to ground produced noise.
- Resonant Frequency- compared to frequency response should be resonant frequency always higher.
- The temperature of Operation – For the correct operation af device a temperature should be between -50 degrees Celsius to 120 degree Celsius. Only with precise installation can be achieved required range.
- Sensitivity – sensitivity of the device should be higher than required. Therefore electrical output signal is high enough to be recognized even for a slight acceleration. This characteristic creates suitable conditions for accurate measurement for any strength of the signal.

- Axis – 2-axis accelerometer is for the most of the industrial applications sufficient enough. Applications where 3D positioning is nedded, a 3-axis accelerometer is required.
- Digital/Analog Output – two possible types of an output, analog output is measured of changing voltages output, digital output is represented by pulse width modulation. (Accelerometer 2011)

3.3 Magnetometer

Magnetometers are devices that measure magnetic fields. The magnetic field of the earth is largest at the poles (~ 60 000 nT) and smallest at the equator (~ 30 000 nT). Magnetic field is measured for different kind of scientific aspects, such as navigation. Magnetometers refer to sensors used for sensing magnetic fields or to systems measuring the magnetic field using one or more sensors. (Preeti 2012)

Magnetometers are divided into two basic types: scalar and vector manometers. A scalar manometer measures the scalar value of the magnetic flux intensity with very high accuracy. A vector manometer measures the magnitude and direction of the magnetic field. (Preeti 2012)

Before selecting a magnetometer for a given application, it is important to consider some of its key characteristics. (GlobalSpec 2018)

- The sample rate is the number of readings given per second. The inverse is the cycle time in seconds per reading. The sample rate is important in mobile magnetometers.
- Bandwidth determines the ability to track quick changes in magnetic field. For magnetometers with no onboard signal processing, bandwidth is determined by the Nyquist limit set by the sample rate. Modern magnetometers may perform smoothing or averaging over sequential samples. achieving a lower noise in exchange for lower bandwidth.

- Resolution is the smallest change in a magnetic field the magnetometer can resolve. A magnetometer should have a resolution a good deal smaller than the smallest change one wishes to observe.
- Absolute error is the difference between the readings of a magnetometer true magnetic field.
- Drift is the change in absolute error over time
- Operating Temperature range in which the device must operate
- The number of axes of simultaneous magnetic field measurement. Single axis sensors are most common; two and three axis sensors are also available.

4 Web API

4.1 General review

An Application Programming Interface (API) is an interface which has a set of functions that allow programmers to access specific features or data of an application, operating system or other services. (Mozilla developer 2018)

Web API is an interface over the web which can be accessed using HTTP protocol, but it is dependent on it, e.g. Constrained Application Protocol (CoAP), an HTTP-like protocol that is common to the Internet of Things. It is a concept and not a technology. Web API can be built on different technologies such as .NET, Java, JavaScript etc. Client does not need to know, which function is called on a server application. Web API implements a set of commands that are built into HTTP, and receiving application has to decide which operation should be called. While HTTPs purpose is to isolate the systems from each other, the systems are looser, and the entire system is less brittle. (Verma 2018)

HTTP defines a set of request methods usually referred to HTTP verbs. Each method implements a different semantics (Mozilla developer 2018):

- GET - The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- HEAD - The HEAD method asks for a response identical to that of a GET request, however, without the response body.
- POST - The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server
- PUT – The PUT method replaces all current representations of the target resource with the request payload.
- DELETE – The DELETE method deletes the specified resource.
- CONNECT – The CONNECT method establishes a tunnel to the server identified by the target resource.

- OPTIONS – The OPTIONS method is used to describe the communication options for the target resource.
- TRACE – The TRACE method performs a message loop-back test along the path to the target resource.
- PATCH - The PATCH method is used to apply partial modifications to a resource.

4.2 REST API

Representational State Transfer (REST) is an architectural style that defines a set of constraints mostly based on HTTP. The concept of REST was introduced and defined by Roye Fieldings in his doctoral dissertation in 2000. Fielding is one of the co-founders of HTTP protocol and therefore, REST, HTTP are bound together. REST is data-oriented architectural style used for uniform and simple access to resources. Data or application states can be required resources. (Malý 2009)

4.2.1 Model

REST implements its own specification and frequently non-written standards, which should be followed. Leonard Richardson developed a 4 levels model shown in Figure 4 explaining REST logic. (Richardson 2010)

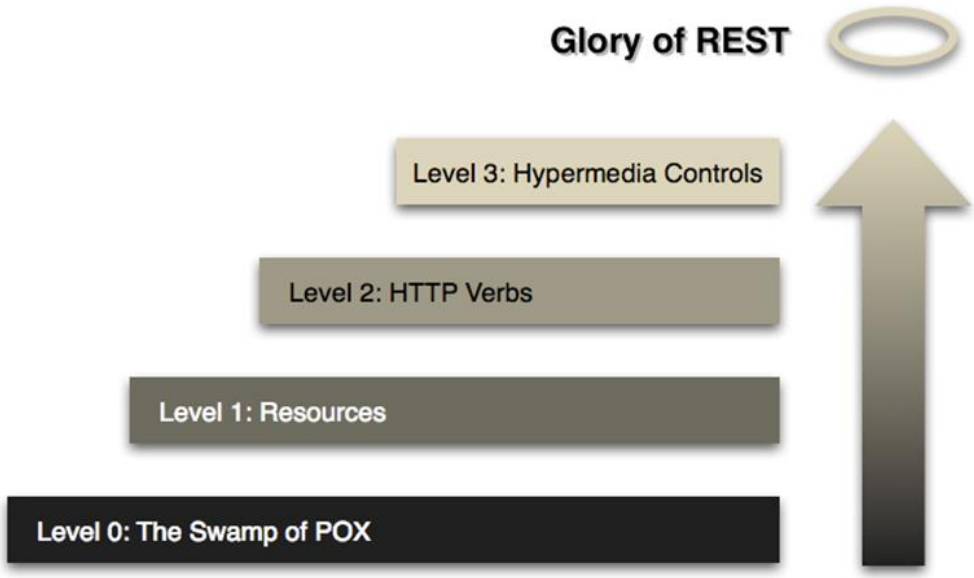

Figure 4 Step to forward REST (Richardson 2010)

Level 0 implementing transport protocol, normally HTTP or other. It will use only one entry point and one kind of method, usually POST. (Richardson 2010)

Level 1. This level uses multiple URIs, where every URI is the entry point to a specific resource. The API is able to distinguish between URIs an invoke appropriate action. From level 1 rather than making all request to a singular service endpoint, the client can call individual resources, e.g. POST /articles – returns all articles, POST /articles/1/comments – returns all comments from the first article. (Richardson 2010)

Level 2. HTTP protocol offers much more methods then POST. All of them are mentioned before in Chapter 3.1 General review. Level 2 uses the HTTP verbs: put, get, post, delete as closely as possible in how they are used in HTTP itself. The meaning of HTTP verbs is (Richardson 2010):

- GET – retrieves requested data. HTTP defines GET as a safe operation, this means that it does not make any significant changes to the state of anything.
- POST – creates a new data.
- PUT – updates a part of existing data Put operation includes path with information about data, which should be changed.
- DELETE – deletes particular data (Richardson 2010)

HTTP included status codes represent success of an operation. The codes are divided into five main categories. The response started with 1xx is informative, 2xx – successful, 3xx – redirecting, 4xx- client error, 5xx-server error. The most frequent are 200 OK, 201 Created related with POST request, 401 Unauthorized client, 404 Not Found. (Richardson 2010)

Level 3 introduces a Hypertext as The Engine of Application State (HATEOAS) The main goal of hypermedia controls is to imform user about next possible steps, that can be executed, and the URI of the resource requires to be managed to be able to do it. Instead of having knowledge of post URI, the hypermedia controls in the response inform how to do it. Reply from POST request contains several hypermedia controls for different thing to do next. The advantage of hypermedia controls is that

the server has the ability to modify its URI scheme without the need of changing the client application. (Richardson 2010)

4.2.2 Design

A software developer should be aware of following six constraints related to REST API design. The following restrictions creates an architecture that act alike pages in browsers on the internet are entered.

The Client-Server limitation is based on the fact that both the client and the server ought to be isolated from one another and be able to expand separately and independently. A Developer has to be capable of changes within the client application without having influence on any kind of data or the database, that can be found on the server. Equivalent guideline ought to be practiced on the server. Modifying database or making changes on the server should not make an impact on the client application. That forms a separation of the concepts, allowing every application spread and be independent from each other. Therefore projects can expand quickly and in an efficient way. (MuleSoft 2018)

"REST APIs are stateless, meaning that calls can be made independently of one another, and each call contains all the data necessary to complete itself successfully." A REST API does not depend on the data stored of the server or sessions to determine what to do with a call, instead of entirely depend on the data that is served in the call itself. Each call consist of the required data e.g. the API key, access token, user ID, etc. "This also helps increase the API's reliability by having all the data necessary to make the call, instead of relying on a series of calls with server state to create an object, which may result in partial fails." Because of the requirements of preserve the application scalable and memory saving, a RESTful API requests client application to store all the necessary states. (MuleSoft 2018)

REST API needs to be constructed to boost the storage of cacheable data. The term of cachable data means, that the response should point out that the data will remain up to a definite time (expires-at), or in cases where data needs to be real-time, that the response should not be cached by the client. By enabling caching in the REST API

results in reducing the number of interactions with API and reducing internal server usage. (MuleSoft 2018)

The uniform interface allows the client communicate to the server in a single language, despite of different backend architecture of each systems. This interface should offer an immutable, standardized types of communication between the client and the server, such as using HTTP with URI resources, CRUD operations, and JSON. (MuleSoft 2018)

A layered system consist of layers, that every layer has a concrete purpose and responsibility. Every layer is separated but at the same time cooperate with each other. Different layers of the architecture working together to build a hierarchy that helps create a more scalable and modular application. (MuleSoft 2018)

Probably the last and the least known of the six constraints is Code on Demand. Only that restriction is considered as a optional. “Code on Demand allows for code or applets to be transmitted via the API for use within the application.” Basically, it construct a smart application that is not anymore entirely rely upon on its own code structure. Code on Demand has struggled with acceptance as Web APIs are consumed across multiple languages and the transmission of code creates security issues. E.g. the directory would have to be writeable, and the firewall would have to accept regularly denied content. (MuleSoft 2018)

4.2.3 JSON

JavaSript Object Notation is a lightweight data-interchange format. It is easy for humans to read and write, and also easy read and generate for computer. ”JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.” These properties make JSON an ideal data-interchange language. JSON is build on two structures (Introducing JSON):

- A set of name/value pairs. In different languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.

- An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These days all the programming languages have feauture to support at least one form of the universal data structure mentioned above.

5 Implementation

5.1 Hardware

For the purpose of measuring data from accelerometer and magnetometer, an embedded device developed by prof. Ing. Juraj Miček, PhD in the department of Cybernetics in Žilinská Univerzita v Žiline was chosen. The device contains following parts:

- 3D accelerometer and 3D magnetometer LMS303DLHC
- Bluetooth HC-05
- 8-bit microcontroller ATmega328
- One Programmable button
- One user LED

The power supply is provided with an external battery. The battery can be charged with an implemented circuit for charging through the micro USB port. Figure 5 shows the mentioned device. The device contains input-output serial pins for test purposes. The developer can easily connect to a serial connection with Bluetooth, check the transmitted date or connect with USB-to-serial converter and flash new firmware on the microcontroller.

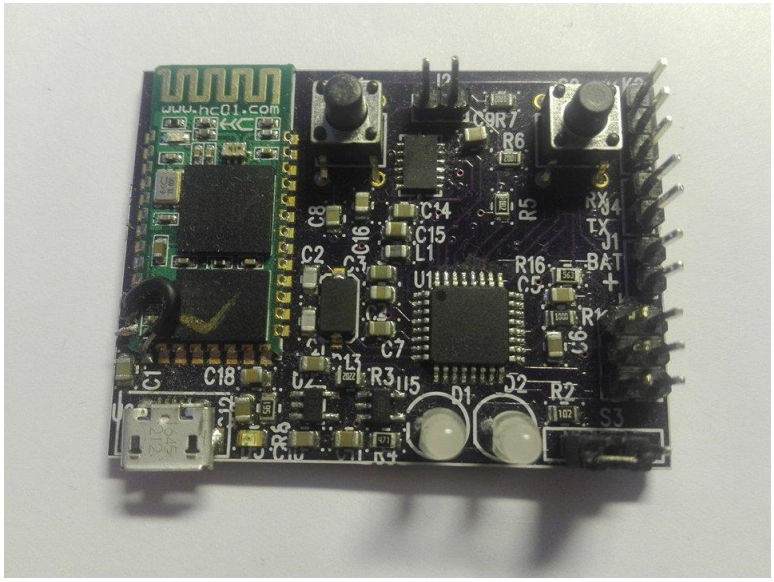

Figure 5 Embedded device

5.1.1 ATmega328 Firmware

The firmware for the microcontroller is consist of two main parts: setup function is always called when the program is triggered and in while loop. In the setup section, all necessary registers values and variables are set up.

At first, peripherals are set up as serial communication with Bluetooth (UART). Figure 6 shows the setup registers required for correct communication with Bluetooth module.

```
void USART0_Init(void)
{
  DDRD |=(1<<PORTD1);          // Tx output
  UBRR0H = (uint8_t)(UBRR_VALUE>>8);  // BAUDERATE 9600
  UBRR0L = (uint8_t)UBRR_VALUE;
  UCSR0C |= (1<<UCSZ01)|(1<<UCSZ00);  //8 bits, parity non, 1 stop
  UCSR0B |= (1<<RXEN0)|(1<<TXEN0)|(1 << RXCIE0);  // enable Rx,Tx, enable Receive interrupt
  UCSR0A |= (1<<U2X0);
  sei(); // Enable the Global Interrupt
}
```

Figure 6 Enable Serial communication

Then follows the setup of I2C communication with an accelerometer. On the microcontroller's side, the interrupt is used for receiving data from the accelerometer. For reading data from the accelerometer, magnetometer, interrupt should be activated first. Then the frequency and mode should be set up, in which should data be sent. Figure 7 shows the setup of the accelerometer.

```
void I2C_init_master(void) // Function to initialize master
{
  TWBR=52;  // Bit rate 100 kHz
  TWSR=(0<<TWPS1)|(0<<TWPS0); // Setting prescaler bits
}
write_register_acc(0x20, 0x47);  // CTR_REG1  50 Hz, normal mode
write_register_mag(0x02, 0x00);  // MR_REG_M,  Continuous-conversion mode
write_register_mag(0x01, 0x20);  //CRB_REG_M, The gain configuration ±1.3
```

Figure 7 I2C configuration

When all components are ready to use, the program starts to execute *while* loop. As in usual microcontrollers, the while loop is endless, and the application will never jump out of it. In the loop, the data transfer through USART to Bluetooth module is received. The Bluetooth module sends immediately data to the connected device.

5.1.2 Data management

The device contains a set of registers used to control its behavior and to retrieve acceleration and magnetometer data. The accelerometer has two options for power mode, normal and low-power. The frequencies can be set up from 1Hz to 400Hz. The high pass filter is implemented into the device and can be enabled for function on interrupt. Three axes for accelerometer can be disabled. The data from each axis is expressed in two 8bit-complements. (LSM303DLHC 2013)

Magnetometer data output rate can be set from 0.75Hz to 220Hz. The temperature sensor is included in the device and can be enabled. Special bit indicates when a new set of measurement is available and ready to read. (LSM303DLHC 2013)

Data from the integrated accelerometer is raw and needed to be calibrated for different usage. Figure 8 shows uncalibrated magnetometer data. For purpose of digital compass magnetometer data needs to be normalized on the center of three-axis coordinate system. Figure 9 shows the magnetometer data calibrated on the center of a 3-axis coordinate.

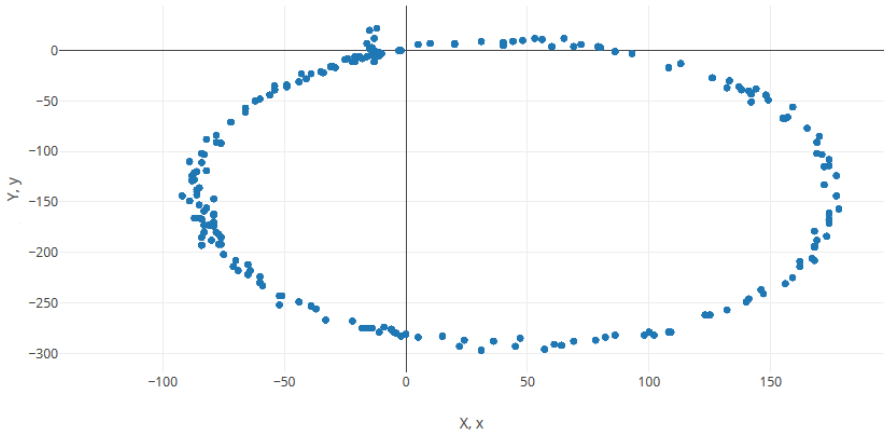

Figure 8 Raw magnetometer data

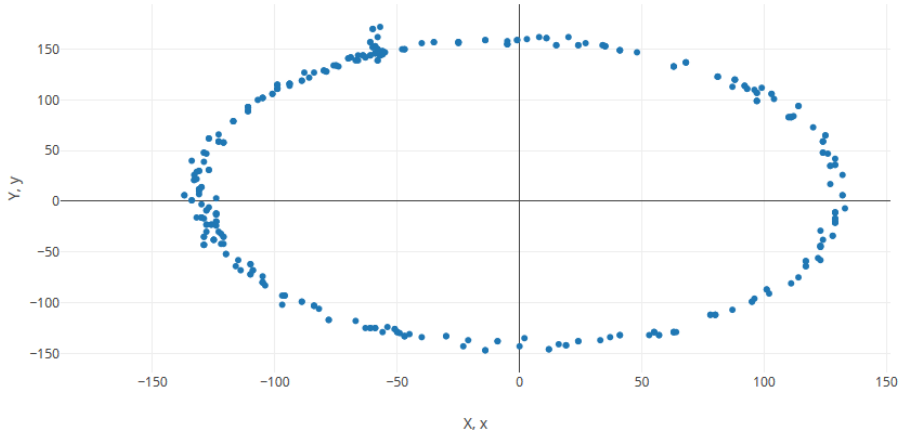

Figure 9 Normalized magnetometer data

5.2 Raspberry Pi

5.2.1 Overview

Raspberry Pi is a credit card sized computer originally designed for education. Raspberry Pi was developed for the Linux operating system, and many Linux distributions now have a version optimized for it. The computing power is lower than in a modern laptop or desktop; however, it is still a complete Linux computer and can provide all the expected abilities at a low-power consumption level. Model Raspberry Pi 3 was chosen for the implementation. (Raspberry 2018)

5.2.2 Threading

Running several threads is similar to running multiple programs at the same time. The usage threads bring following benefits. Multiple threads within a process share the same data space with the main thread and can, therefore, share information or communicate with each other more easily than if they were separate processes. Threads do require less memory over the processes and they are sometimes called light-weight processes. A thread has a beginning, an execution body, and end. A thread can be temporarily interrupted while other threads are running. (Python multithreading)

The thread constructor should always be called with keyword arguments. threading.Thread(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs={}*, ***, *daemon=None*) All possible arguments are (Threading 2018):

- *Group* should be none, this argument is reserved for future extension when a *ThreadGroup* class is implemented.
- *Target* is the callable object to be invoked by the *run()* method. Defaults to None, meaning nothing is called.
- *name* is the thread name. By default, a unique name is constructed of the form "Thread-*N*" where *N* is a small decimal number.
- *args* is the argument tuple for the target invocation. Defaults to ().
- *kwargs* is a dictionary of keyword arguments for the target invocation. Defaults to {}.
- If not None, *daemon* explicitly sets whether the thread is daemonic. If None (the default), the daemonic property is inherited from the current thread. (Threading 2018)

Figure 10 shows creation a thread in the program as the argument is passed into worker dictionary with recorded data.

```
t = threading.Thread(target=worker, args=(dataDictionary,))
t.start()
```

Figure 10 create a new thread

The Thread class represents an activity that is run in a separate thread of control. Once a thread object is created, its activity must be started by calling the thread's *start()* method. This invokes the *run()* method in a separate thread of control. In program is used function *worker(data)* as target function. Figure 11 shows the body of worker function. (Threading 2018):

```
# thread function
def worker(data):
    logging.debug('Starting')
    request = Request()
    request.sendPost(data)
    logging.debug('Exiting')
```

Figure 11 thread function

5.2.3 Functionality

Raspberry Pi is used as a gateway between embedded device and server. The whole application is written in Python. Bluetooth communication protocol is implemented for the purpose of communication with the embedded device. The communication channel is established for transmitting data and commands. Capturing data from the device can be easily controlled remotely. The speed of communication is set for 9600 Baud. The data is received as a stream. Every line contains six parts from every axis. The data is separated and added to the data structure. The data structure is a simple object containing variables for all axes and timestamp. The timestamp is added to data immediately after receiving. Figure 12 shows used data structure.

```
class Measurement:
    x = []
    y = []
    z = []

    def __init__(self):
        self.x = []
        self.y = []
        self.z = []

class DataModule:
    title = 0
    acc = Measurement()
    mag = Measurement()
    timestamp = 0

    def __init__(self):
        self.title = "title"
        self.acc = Measurement()
        self.mag = Measurement()
        self.timestamp = time.time()
```

Figure 12 Data structure

The crucial part every application which recording data from all kind the sensors is a frequency. This depends on the character of an application, and how high frequency should be used. The required frequency can be achieved by changing the accelerometer register value or application flow with waiting.

The received data is collected into a dictionary. The size of a dictionary can be set according to requirements. The final dictionary is transformed to JSON file with an array of objects. This file is sent to the server as HTTP post request. For post request, Python requests library is used, which can handle post request and analyze the response from the server. Threading is used for sending post requests. A single thread is created, and it handles post request through its life cycle.

5.3 Server application

5.3.1 Overview

The server application is written in NodeJS. NodeJS is an open-source JavaScript run-time environment that executes JavaScript code. The main function of the application receives measured data in JSON format file at any time when the device is active, stores it in a database and serve data to other applications. The application implements a calculation compass heading logic and it is ready to be extended with the web application. Figure 13 illustrates an application workflow from embedded device to users web application.

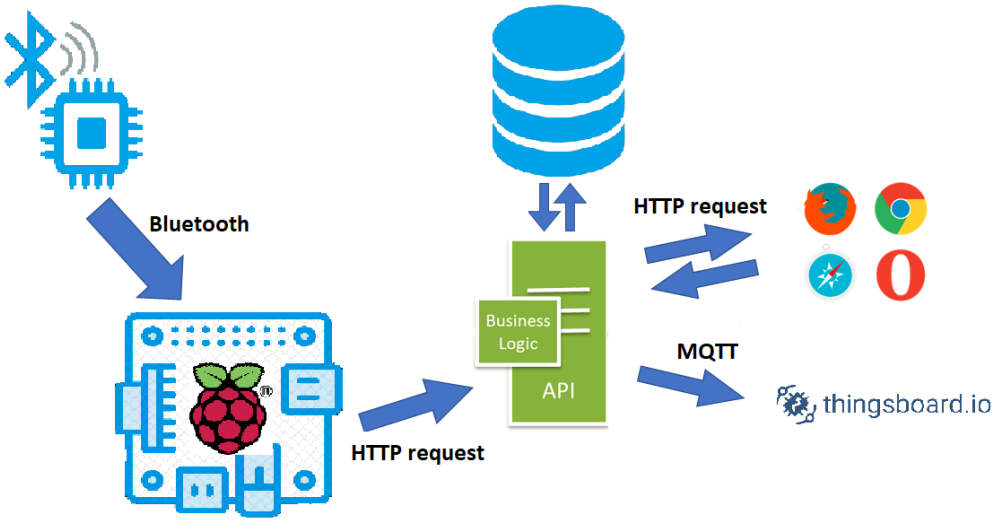

Figure 13 application schema

5.3.2 Functionality

REST API is implemented with espress.js. Express.js is a minimalistic web framework for Node.js. The application implemented HTTP routes for all necessarily requests from Python application and from the web application. Basically, the Python application uses only POST request. Server application does not transmit any data to Python application. Necessarily data needed by Python application can be requested via *get* HTTP request.

After the application is started HTTP server is created. The server runs on localhost for test purposes. For production, a public IP with a port for the server should be set up. Next, the server tries to connect to the database. Without a successful connection to the database, the server does not have a storage for data. The simple route as *get* request is the last thing set up, which returns to the server when an HTTP request without any parameters is called. After the initialization, the server is ready to handle requests from clients.

The data received from Raspberry is encapsulated into JSON file. The server first checks if the file contains a title and measurements. The title contains a value of the actual measurement. Without those two values, the file invalid is received and will not be more processed. The client receives information about an occurred error. After valid checking data is parsed into the data model and saved to the database. Figure 14 shows the above-mentioned function.

```
//create a collection of records
exports.createList = function(req, res) {
  if(!req.body.title) {
      return res.status(400).send({message: "Title cannot be empty"});
  }
  let array = req.body.meranie;
  if(array.length == 0) {
    return res.status(400).send({message: "No measured data"});
  }
  let notelist = new NoteList({title: req.body.title || "Untitled Note",
                               timestamp: req.body.timestamp,
                               children: [] })
  for (let i = 0; i < array.length; i++) {
    notelist.children.push({acc: array[i]["acc"], mag: array[i]["mag"], timestamp: array[i]["timestamp"]});
  }
   let lastItem = notelist.children[array.length - 1];
  // save received data collection to DB
  notelist.save(function(err, data) {
      if(err) {
          console.log(err);
          res.status(500).send({message: "Some error occurred while creating the Note."});
      } else {
          res.send(data);
      }
  });
}
```

Figure 14 Post request from raspberry

Follows gets requests. The first one *findAll* returns all data from the database without any restrictions. The second function *findOne* is looking just for a specific record. The criterion for the record is time. Time is stored in every collection as a timestamp. The server notifies the client whether with retrieved data or with a specific error code. Figure 15 shows the above-mentioned function.

```
exports.findOne = function(req, res) {
  console.log("findOne", req.params.time);
  //find a collection from measurement with specific timestamp
  let collection = {};
  NoteList.
    find({'title':'meranickoRanicko', 'timestamp': req.params.time},
      function(err, note) {
        if(err) {
          return res.status(500).send({message: "Error retrieving note with timestamp: "+ req.params.time});
        }
        if(!note) {
          return res.status(404).send({message: "Note not found with timestamp: " + req.params.time});
        }
        res.send(note);
    });
};
```

Figure 15 Get request findOne from client

5.3.3 MongoDB

MongoDB is an open-source document database designed for the ease of development and scaling. The data is stored in flexible, JSON-like documents. This type of storage allows changing the data structure from one document to another over time. The document model maps the project objects, as can be seen, in Figure 16, in application code, thus, making it easy for the data to work with. Ad hoc queries, indexing, and real-time aggregation provides powerful ways to access and analyze data. (MongoDB 2018)

Data in MongoDB has a flexible schema. MongoDB uses collections instead of tables. Unlike SQL databases, table schema does not need to be declared before inserting data. This flexibility facilitates the mapping of documents to an entity or an object. It is recommended to embed related data in a single structure or document, e.g. data from an embedded device contains three values from accelerometer and magnetometer. There is no need to place separate data into more documents. They are embedded into one document. (MongoDB 2018)

```
var NoteSchema = mongoose.Schema({
    acc: {
      aX: Number,
      aY: Number,
      aZ: Number
    },
    mag: {
      mX: Number,
      mY: Number,
      mZ: Number
    },
    timestamp: Number,
}, {
    timestamps: true
});

var NoteListSchema = mongoose.Schema({
    title: String,
    timestamp: Number,
    children: [NoteSchema]
}, {
    timestamps: true,
    collections: 'user1data'
});
```

Figure 16 MongoDB model

5.3.4 Compass formula

A compass heading can be determined by using just the X and Y component of the earth's magnetic field, i.e. the directions planar with the earth's surface. The

magnetometer is held flat in an open area and the Y and Y magnetic readings noted. These readings vary as the magnetometer is rotated in a circle as shown in Figure 17. The maximum value of Hx and Hy depend on the strength of the earth's field at that point. The magnetic compass heading can be determined in degrees from the magnetometer's x and y readings by using the function shown in Figure 18.

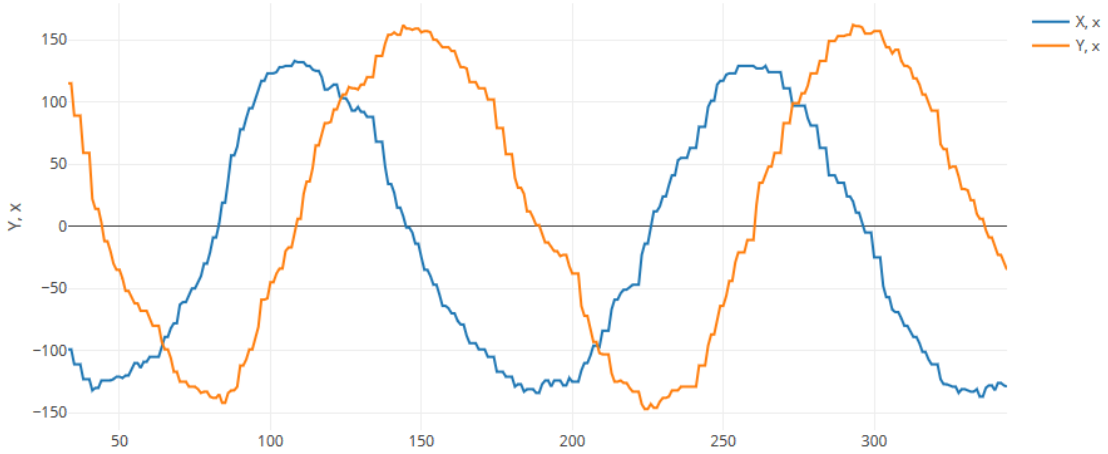

Figure 17 magnetometer horizontal rotation

```
Compass.prototype.calculateDirection = function(data) {
  if (data.hasOwnProperty('x')) {
    let heading = (Math.atan2(data.y, data.x)*180)/Math.PI;
    if (heading < 0) {
      heading += 360;
    }
    return heading;
  }else {
    return 0;
  }
}
```

Figure 18 compass heading function

5.4 ThingsBoard platform

5.4.1 Overview

The purpose of IoT device is to connect with another IoT device and application to relay information over the internet and bring it to the end user. IoT platforms fill the gap between the device and data network. IoT platforms connect hundreds of sensors, and using backend application makes sense of the plethora of data generated by devices. ThingsBoard IoT platform was chosen for this project. ThingsBoard is an open-source IoT platform for data connection, processing, visualization and device management. The devices can be connected via standard IoT protocols as MQTT, CoAP, and HTTP. Cloud and on-premise deployment are available. User interface and REST APIs are provided to manage multiple entity types and their relations in the IoT application. Each entity has access to attributes, telemetry data, and relations. The supported entities are Users, customers, devices, assets, alarms, dashboards, etc.

5.4.2 Device attributes and telemetry

ThingsBoard provides the ability to assign attributes to your entities and manage them. Attributes create key-value pairs. The flexibility and simplicity of the key-value format attributes allow easy integration with almost any IoT device. Attributes are static or semi-static data associated with entities. All attributes may be used in Rule Engine components. With the rule, the engine allows sending an email when an attribute is changed, an alarm is created when telemetry exceeds a certain threshold; the telemetry data is forwarded to the external RESTful server etc.

Device attributes are divided into three main categories (ThingsBoard docs 2018):

- *Server-side*: attributes are reported and managed by the server-side application. The device application does not have access to these attributes. Any ThingsBoard entities support server-side attributes. Some secret data may be used by ThingsBoeard which cannot be visible and accessible by the device.

- *Client-side;* attributes are reported and managed by the device application. For example, current software/firmware version, hardware specification.
- *Shared:* attributes are reported and managed by the server-side application but visible for device application. (ThingsBoard docs 2018)

ThingsBoard provides API to the device application, which defines the follows functions. The client can upload attributes to the server, requests for clients and shared attributes from the server and subscribe the updates of shared attributes.

ThingsBoard consists of core services and pluggable modules. The telemetry module is one of them. Telemetry data is time-series data points available for storage, querying and visualization, e.g. temperature, speed, battery status, compass heading. Since telemetry is crucial for visualization purposes in a dashboard, it is configurable by the system administrator on the system level. Data can be forwarded to the database directly. NoSQL Cassandra database and SQL database are supported and can be used for internal data storage. MQTT protocol was used to update device telemetry in the project. Figure 19 shows the function responsible for publishing a new device telemetry. (ThingsBoard docs 2018)

```
  function uploadStats() {
    let data = cp.updateTelemetry();
    console.log('Device %s. Publishing OS info & stats: %s',cp.serialNumber, JSON.stringify(data));
    client.publish(telemetryTopic, JSON.stringify(data));
  }

}
```

Figure 19 Update device telemetry

Data visualization is one of the main functions of ThingsBoard. IoT dashboards are used for visualizing the device data. Each dashboard may contain multiple dashboard widgets and visualize data from different devices. The dashboard can be created manually or automatically when a new device is assigned to a customer. The dashboard may contain a widget from the library, or the widget can be created and

uploaded by the user. ThingsBoard library contains widgets for analog and digital gauges for the latest real-time values. Customizable charts are used for visualization of historical data points, map widgets for real-time tracking and the latest position of a device on Google or OpenStreet maps. GPIO control widget allows sending GPIO toggle commands to the device.

Analog compass widget was used for visualizing the latest values from the device in the project. The widget implements a simple pointer for the purpose of heading visualization. The widget requires a number in the range from 0 to 360, which refers to the actual direction. The used widget can be seen in Figure 20.

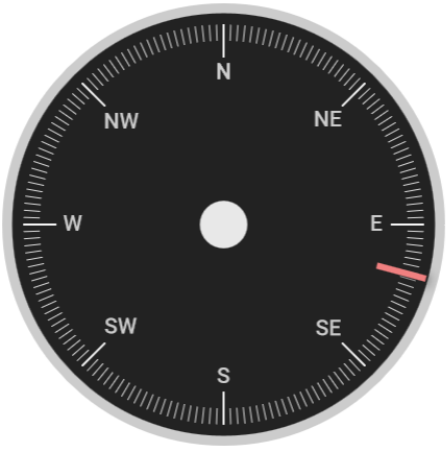

Figure 20 compass widget

6 Summary

The main goal of this paper was to create an application consisting of three separate projects. In the projects, three different programming languages were used. The purpose of the application was to provide monitoring an embedded device collect measured data and visualize them.

In the beginning, I was challenged with new technologies such as NodeJS, JavaScript, and client-server architecture. It took me few weeks to make a research about these new technologies and how to use them. Well written documentation and plenty of examples helped me to understand the principles.

An essential part of the project was to understand the measured data. Getting to know how to read data from the device and what it represents was crucial. Noise always affects recorded data. Both accelerometer and magnetometer are sensitive sensors and noise should be cut out to reach valuable results. The accelerometer has an anti-aliasing filter that is often used to soften unwanted abnormalities. In the project, the application used data from the magnetometer to calculate digital compass. The magnetometer retrieved 3-axis values. For correct calculation, the data should be normalized to the center of the three axes coordinate as referred to in Figure 9.

I was able to successfully develop an application, which measures and visualizes compass data in real time. A gyroscope should be implemented into the device to get more valuable information about the device. With the gyroscope, it would be possible to calculate an angle of the device.

Possible usage of this device with the implemented gyroscope is to monitor a truck trailer and predict a possibly dangerous situation when the trailer is about to be overturned. Another use case is to monitor a barbell of weightlifters. During the lifting, the barbell should have a straight trajectory. The device could be used for training purposes and prevention from dangerous and incorrect lifting.

During the implementation, I ran out into some limitations. The device always has to be in a Bluetooth perimeter from Raspberry. This limitation could be avoided with

changing the technology from Bluetooth to a technology with a longer range, e.g. LoRa. The second limitation appeared when I tried to implement backward communication from the server-side application to Python script. Because of the client-server architecture of REST API, only the client can request data and the server does not have a possibility to invoke an action without a request from the client. This creates an issue when a client needs to send a request often in intervals to ask if the data recording should be triggered on.

I gained valuable skills from client-server architecture. I learned about RESTful design and how to implement it with different technologies. During the development of data management functionalities, many tutorials and scientific researchers helped me to understand the data.

In this paper, the reader finds coherent information about developing a full-stack project. The server application can be used for creating a web application of any kind to request collecting data from sensors stored them in a server.

References

Accelerometer 2011. What is an Accelerometer? Accessed on 9.5.2018. Retrieved from http://www.instrumentationtoday.com/accelerometer/2011/08/

ARM Articles 2008. What is the difference between a von Neumann architecture and a Harvard architecture? Accessed on 11.5.2018. Retrieved from http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka3839.html

Basheer N. 2013. Architecture of 8031/8051 Microcontroller. Accessed on 18.4.2018. Retrieved from https://mediatoget.blogspot.fi/2013/03/architecture-of-80318051-microcontroller.html

Brain M. 2000. How microcontrollers works. Accessed on 22.3.2018. Retrieved from https://electronics.howstuffworks.com/microcontroller1.htm

Embedded system. Accessed on 22.3.2018. Retrieved from https://www.tutorialspoint.com/embedded_systems/es_microcontroller.htm

EnTcians, 2013. A technical blog of Students of E&TC, SITRC, Nashik. Accessed on 23.3.2018. Retrieved from http://electro-mate.blogspot.fi/2013/07/classification-of-microcontrollers.html

GlobalSpec 2018. Magnetometer specification. Accessed on 11.5.2018. Retrieved from https://www.globalspec.com/SpecSearch/SearchForm/motion_controls/orientation_position_sensing/compasses_magnetometers

Hemingway B. 2015. Software for Embedded Systems. Accessed on 9.5.2018. Retrieved from https://courses.cs.washington.edu/courses/cse466/15au/pdfs/lectures/MEMS%20Sensors.pdf

Introducing JSON. Accessed on 17.5.2018. Retrieved from https://www.json.org/

LSM303DLHC, 2013. STMicroelectronics. Accessed on 7.5.2018. Retrieved from http://www.st.com/resource/en/datasheet/DM00027543.pdf

Malý M. 2009. REST: an architecture for web API. Accessed on 24.4.2018. Retrieved from https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

MongoDB 2018. MongoDB documentation. Accessed on 11.5.2018. Retrieved from https://docs.mongodb.com/

MuleSoft 2018. Understanding REST API Design. Accessed on 11.5.2018. Retrieved from https://www.mulesoft.com/resources/api/what-is-rest-api-design

Mozilla developer. 2018. HTTP request methods. Accessed on 24.4.2018. Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

MCU Programming. 2017. Accessed on 23.2.2018. Retrieved from https://www.renesas.com/en-eu/support/technical-resources/engineer-school/mcu-programming-peripherals-01-gpio.html

Omega company, 2003. Accessed on 9.5.2018. Retrieved from https://www.omega.com/prodinfo/accelerometers.html#learn

Preeti J, 2012. Magnetometers. Accessed on 9.5.2018. Retrieved from https://www.engineersgarage.com/articles/magnetometer

Python multithreading. Accessed on 8.5.2018. Retrieved from https://www.tutorialspoint.com/python/python_multithreading.htm

Raspberry 2018. Raspberry Pi documentation. Accessed on 11.5.2018. Retrieved from https://www.raspberrypi.org/documentation/

Richardson L, 2010. Richardson Maturity Model. Accessed on 22.4.2018. Retrieved from https://martinfowler.com/articles/richardsonMaturityModel.html

Rouse M. 2016. Embedded systems. Accessed on 21.3.2018. Retrieved from http://internetofthingsagenda.techtarget.com/definition/embedded-system

Texas Instruments TMS1000 microcontroller family. 2001. Accessed on 23.3.2018. Retrieved from http://www.cpu-world.com/CPUs/TMS1000/index.html

The 8051 Block Diagram, 2011. Accessed on 18.4.2018. Retrieved from https://aninditadhikary.files.wordpress.com/2011/01/8051blockdiagram.png%3Fw%3D593

ThingsBoard docs, 2018. Accessed on 10.5.2018. Retrieved from https://thingsboard.io/docs/user-guide/mqtt-over-ssl/

Threading, 2018. Accessed on 8.5.2018. Retrieved from https://docs.python.org/3/library/threading.html

Understanding the I2C bus, 2015. Accessed on 23.3.2018. Retrieved from http://www.ti.com/lit/an/slva704/slva704.pdf

Verma S. 2018. APIs versus web services. Accessed on 24.4.2018. Retrieved from https://blogs.mulesoft.com/dev/api-dev/apis-versus-web-services/

Wikipedia Embedded system. Accessed on 18.4.2018. Retrieved from https://en.wikipedia.org/wiki/Embedded_system

Wikipedia Microcontrollers. Accessed on 15.5.2018. Retrieved from https://en.wikipedia.org/wiki/Microcontroller

Wikipedia USART. Accessed on 17.4.2018. Retrieved from https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter

Wikipedia, 2002. Magnetometer. Accessed on 5.5.2018. Retrieved from https://en.wikipedia.org/w/index.php?title=Magnetometer&action=history&year=2004&month=-1&tagfilter

Wikipedia SPI. Accessed on 17.4.2018. Retrieved from https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus