COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND
INFORMATICS

# INTERNET OF THINGS IN AUTOMOTIVE INDUSTRY

**Bachelor's thesis**

**2016** **Dávid Kőszeghy**

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND
INFORMATICS

# INTERNET OF THINGS IN AUTOMOTIVE INDUSTRY

**Bachelor's thesis**

| | |
|---|---|
| Study Programme: | Applied Computer Science |
| Field of Study: | 2511 Applied Informatics |
| Department: | Department of Applied Informatics |
| Supervisor: | RNDr. Jozef Šiška |
| Advisor: | Mgr. Anton Pytel |

**Bratislava 2016** **Dávid Kőszeghy**


Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

# THESIS ASSIGNMENT

| | |
|---|---|
| **Name and Surname:** | Dávid Kőszeghy |
| **Study programme:** | Applied Computer Science (Single degree study, bachelor I. deg., full time form) |
| **Field of Study:** | Applied Informatics |
| **Type of Thesis:** | Bachelor´s thesis |
| **Language of Thesis:** | English |
| **Secondary language:** | Slovak |

**Title:** Internet of Things in automotive industry

**Aim:** Aim of this bachelor thesis is to explore possibilities of combining Internet of Things, Big Data and automotive industry, e.g. cars. Solve how to connect cars to the Internet. Explore what data can be collected with connected car. Design how to structure and collect data. Analyze capabilities of collected data. Develop an application which will use the collected data. Create a Proof-of-Concept device showing the capabilities of combining Internet of Things with vehicles. Present use-case of the designed device and possible future development.

**Keywords:** Internet of Things, Connected Cars

| | |
|---|---|
| **Supervisor:** | RNDr. Jozef Šiška, PhD. |
| **Department:** | FMFI.KAI - Department of Applied Informatics |
| **Head of department:** | prof. Ing. Igor Farkaš, Dr. |

**Assigned:** 28.10.2015

**Approved:** 29.10.2015 doc. RNDr. Damas Gruska, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

## Declaration of Authorship

I confirm that this Bachelor's thesis is my own work and I have documented all sources and material used.

Bratislava May 26, 2016 .....

## Acknowledgements

I would like to thank my supervisor RNDr. Jozef Šiška for providing valuable feedback and ideas. I would also like to thank my advisor Mgr. Anton Pytel for continuous feedback and insight. Special thanks goes to the Softec, spol. s r.o. without which this collaboration would not be possible.

# Abstrakt

Počítač pripojený k autu pomocou OBD-II portu vie poskytnut rôzne možnosti zbierania dát. Aplikovaním konceptu Internetu Vecí táto bakalárska práca ponúka riešenie integrácie dát ukázaním aké možnosti a služby vieme ponúknuť, keď pripojíme auto na Internet. Po preštudovaní tejto práce by každý so znalosťami z oboru informatiky mal vedieť využiť ponúkané možnosti auta pripojeného na Internet podľa jeho alebo jej potrieb.

**Kľúčové slová:** *Automobilový priemysel, Internet Vecí, Veľké Dáta, Vstavané zariadenie, OBD-II port*

# Abstract

Computer connected to OBD-II port of a vehicle can provide miscellaneous options for data gathering. With Internet of Things concept in mind this bachelor thesis provides solution of data integration by showing the possibilities and services connected vehicle can provide. After a thoughtful study of our work anybody with computer science background will be able to build a device which will be capable of take advantage of connected car based on his or her needs.

**Keywords:** *Automotive industry, Internet of Things, Big Data, Embedded device, OBD-II port*

# Contents

# Introduction

Most of us who watched Knight Rider as a child expected that by 2015 we would be driving self-aware cars like KITT - cars that would drive us without problems from point A to point B while entertaining us. However, this is only partially true, we have successful tries at completely autonomous cars(e.g Google car) but for ordinary customer they are not available. What is available is connected cars. Connected cars provide the possibility of Internet-based transfer of information. This can be obtained either with embedded devices in car or smartphone. Multiple car manufacturers sell connected cars with different OEM(Original Equipment Manufacturer) devices or applications. These OEMs offer various services which are provided to you when you purchase their car, but there is a small problem. Often these OEM devices have been designed to work with applications provided from car manufacturer and it is not possible to use other applications to gather information from the car. They are not configurable to the extent as smartphones have applications. It is either getting used to the OEM solution or buying whole solution from somebody else. It is like we are stuck in the 90s, where software offered had to come on this particular hardware and it could not be chosen otherwise. Imagine you would buy smartphone with predefined set of applications and it is not possible to change them or add new. To sum it up, new connected cars lack easy configuration of information gathering, using this information according to wishes of customer, not car manufacturer. So in our thesis we will provide solution for these problems, build a device which will be capable of collecting information and act based on it. By following our thesis insight will be gained into how to connect car and what possible services might be obtained. In first chapter we will establish some common ground on upon which we can build foundations of this thesis. In next step we will describe the specification of problem, design a solution for it, and show implementation of the solution to this given task.

# 1. Overview

In this chapter overview of technology and already available solutions is provided. We will look into how car electric systems works, what embedded devices can be bought on the market or what IoT means.

## 1.1 Solutions available

In this section we will provide an overview of existing solutions which are similar with the goal of this bachelor thesis.

### 1.1.1 Automatic

“The Automatic car adapter plugs into just about any car’s standard diagnostics (OBD-II) port. It unlocks the data in your car’s on-board computer and connects it to your phone via Bluetooth wireless.”[8]
Automatic turns almost any car into a connected car. By pairing Automatic’s adapter and apps for iPhone, Android, and web, drivers are able to enhance their driving experience with a host of connected services on the Automatic platform. Automatic helps to diagnose engine trouble, also provides accelerometer which measures car orientation to detect serious collision. By using audio signals Automatic is able to provide feedback to the driver. As Automatic connection is wireless, all data is encrypted by using 128-bit AES encryption.[6] More detailed hardware specification can be found in the Appendix A. This particular solution is proprietary and can be bought in the U.S.

### 1.1.2 VI Monitor

The VI Monitor works by reading the data stream straight form vehicle electronic control unit via On-Board Diagnostics port (OBD-II). The VI Monitor has also a 3.5" touch screen display and G-sensor. It has an advanced diagnostics tool with the ability to view and reset engine fault codes and comes with sophisticated comparison software. Main difference

comparing Automatic and VI Monitor is in set of tools, VI Monitor allows for precision braking and acceleration tests by using accelerometer and car data.[46] For more detailed overview of capabilities of VI Monitor consult Appendix B.

## 1.2 Internet of Things

The Internet of Things (IoT) is a global infrastructure concept to further informatization of society, enabling advanced services by interconnecting things based on available technologies.
The IoT technology aims to build a set of networks in which each object is connected. In the IoT, all objects have storage and computing power[35]. Through identification, data capture, communication and processing capabilities, the IoT makes full use of things to offer services to all kinds of applications.
Whilst IoT is a hot topic in the industry it is not a new concept. It was initially put forward by Mark Weiser in the early 1990s. This concept is opening up huge opportunities for both the society and individuals. However, it also involves risks and undoubtedly represents an immense technical and social challenge.[13]

### 1.2.1 Concept

"Things", in IoT refer to a various devices. From hardware level they are all designed to do different things, collect data from various environments and sources. However, in software aspect they behave more generally, in simplistic form it is a thing which can report data and act upon them.
Objectification is important, because then it can be combined with many things to work together and communicate between them. Current market example could be the smart thermostat systems combining washer/dryers that use Wi-Fi for remote access.

### 1.2.2 Benefits

IoT is generating a lot of interest in a wide range of industries. Here are a few examples of some significant early adopters:

- In the healthcare field, medical device manufacturer Varian Medical Systems is seeing a 50 percent reduction in mean time to repair their connected devices.[30] With IoT,

Varian reduced customer service costs by $2,000 for each problem resolved remotely, with 20 percent fewer technician dispatches worldwide.

- Italian tire maker Pirelli is using IoT to gain valuable insights about the performance of its products in near real time.[28] The company is using an analytics platform to manage the huge amounts of data gathered directly from sensors embedded in the tires in its Cyber Tyre range. The system allows the pressure, temperature, and mileage of each tire to be monitored remotely. By keeping these factors in range, fleet managers can have a significant impact on fuel economy and safety. In a trial covering nearly 10 million miles, Cyber Tyres saved the equivalent of $1,500 per truck per year[7].

- Ford Motor Company's Connected Car Dashboards program collects and analyzes data from vehicles in order to gain insights about driving patterns and vehicle performance. The data is analyzed and then visualized graphically using a big data platform. Among the goals are better vehicle design and improved safety for occupants.[28]

- US-based HydroPoint Data Systems is using the loT in its WeatherTRAK application to enable landscape irrigation systems, sensors, and computers to communicate autonomously online to identify leaks and other potential system problems. [28]

IoT has the potential to transform the way companies make products, track goods and assets in the supply chain, monitor the performance of systems in the field, provide security for employees and facilities, and provide services to customers. Clearly, it's enabling transformation in both the private and public sectors.
"I firmly believe that there is not a single industry that will not benefit from IoT." – said Vernon Turner, Senior Vice President of Research and IoT Executive Lead at International Data Corp.(IDC). IoT is also changing the way businesses impact society and the environment. "Overall, the world needs better and more sustainable ways to live," said Stephen Miles, research affiliate at the Center for Biomedical Innovation at the Massachusetts Institute of Technology. "To accomplish this, companies need better, more holistic models that capture a complete picture of what is happening so that they can better access and optimize these systems."

## 1.3 $I^2C$

Inter-Integrated Circuit ($I^2C$) protocol is a very popular serial protocol, mainly because it is cheap, simple and allows to connect multiple devices. Whole $I^2C$ bus can be made of only 2 lines: SDA and SCL. SDA is a data line which is carrying data from one device to another and SCL is clock line which main purpose is to synchronize all the devices connected using the $I^2C$ bus. The SDA and the SCL lines are mandatory for all devices which support connection using $I^2C$ bus[20]. The block diagram representation of an $I^2C$ bus is as shown in 1.1

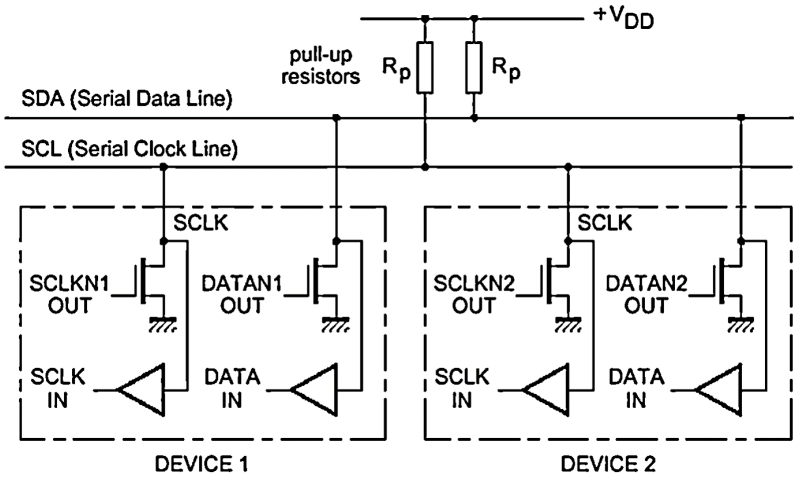



**Figure 1.1:** $I^2C$ bus.[1]

### 1.3.1 Bus signals

Both signals (SCL and SDA) are bidirectional. They are connected through resistors to a positive power supply voltage. This means both lines are high when the bus is free. Activating the line means pulling it down (wired AND). The number of the devices on a single bus is almost unlimited – the only requirement is that the bus capacitance does not exceed 400 pF. Because logical 1 level depends on the supply voltage, there is no standard bus voltage.[19]

### 1.3.2 Hierarchy

$I^2C$ devices are can be registered as master or slave. Masters initiate a message and slaves respond to a message. A master can have multiple slaves and any device can be master-only, slave-only, or switch between as provided in image 1.2.

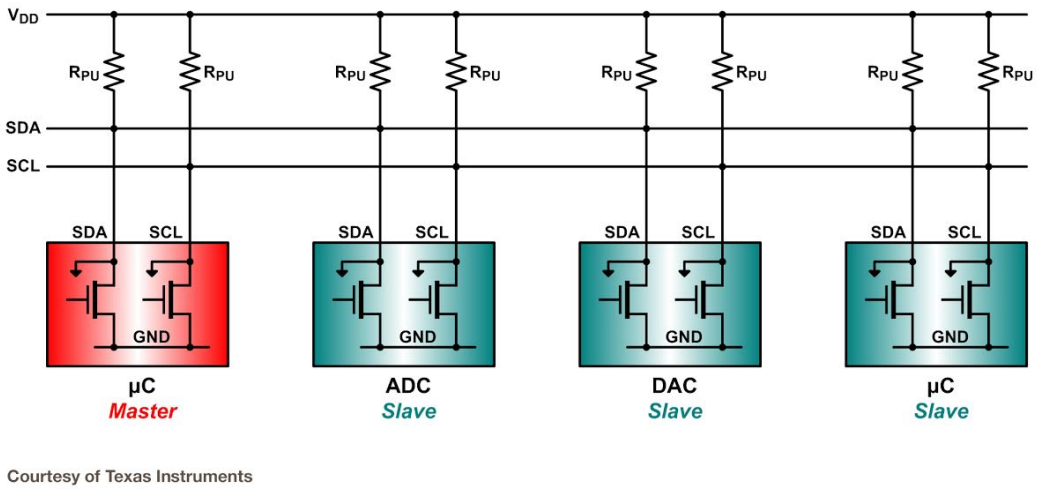



**Figure 1.2:** Multiple $I^2C$ devices connected on the bus.[24]

## 1.4 Single-board computer

The Single-board computer (SBC) is complete functional computer built on single circuit board. It has microprocessor, memory, input/output and other features depending on the model and manufacturer. SBC are used for educational purposes, embedded solutions, development research/systems and sensor monitoring systems which are used in many applications such as greenhouses, ecology studies, automobiles, robots, and medical devices. These systems are essentially used to study environment data like temperature, light, humidity, or air pressure. These sensor measurements are used in manual and automation control based systems[5].

### 1.4.1 Platforms

For making a informed decision which SBC to use for this thesis, we provide short overview of chosen manufacturers and their platforms for embedded devices.

#### 1.4.1.1 Raspberry Pi

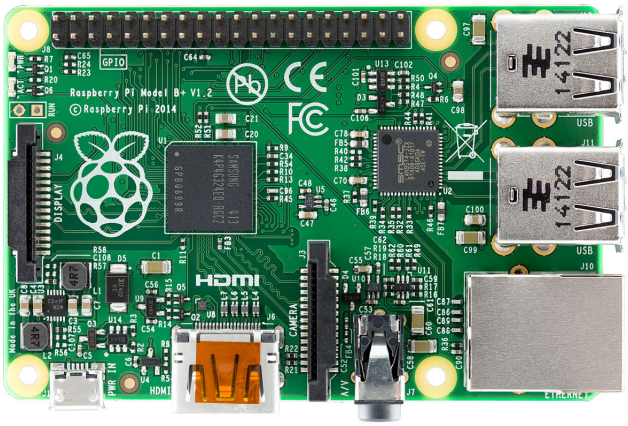


**Figure 1.3:** Raspberry Pi Model B+ v1.2 [27]

Raspberry Pi is a credit-card sized mini computer as shown at Figure 1.3. It is a low cost solution to many projects, and its main goal is to enable people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It is fully capable of running operating system based on ARM architecture. It has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras.[16]. Raspberry Pi foundation developed various models with similar hardware specifications. At the time of writing this thesis the most advanced model was Raspberry Pi 3. Hardware specifications are shown at Table 1.1. Best addition to previous version was on-board wireless chip, which allowed the device to be connected to the wireless network.

| | |
|---|---|
| **SoC**: | Broadcom BCM2837 |
| **CPU**: | 4x ARM Cortex-A53, 1.2GHz |
| **GPU**: | Broadcom VideoCore IV |
| **RAM**: | 1GB LPDDR2 (900 MHz) |
| **Networking**: | 10/100 Ethernet, 2.4GHz 802.11n wireless |
| **Bluetooth**: | Bluetooth 4.1 Classic, Bluetooth Low Energy |
| **Storage**: | microSD |
| **GPIO**: | 40-pin header, populated |
| **Ports**: | HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0 |
| | Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI) |

**Table 1.1:** Raspberry 3: Hardware specification

#### 1.4.1.2 The Beagles

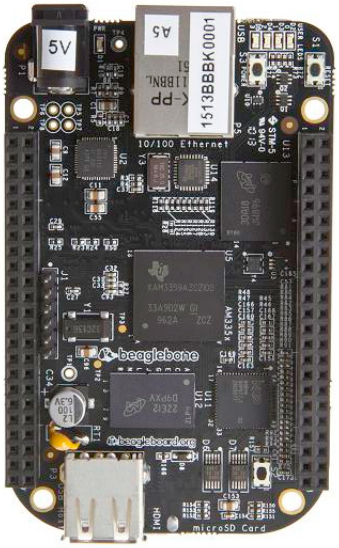



**Figure 1.4:** BeagleBone Black[9]

The BeagleBoard Foundation is a US-based non-profit corporation existing to provide education in and promotion of the design and use of open-source software and hardware in embedded computing. Providing a platform for the owners and developers of open-source software and hardware to exchange ideas, knowledge and experience.[23]. The Beagles are open-hardware and open-software computers, that can be used for numerous projects, same as Raspberry Pi. In time of writing this thesis most popular model among the Beagles was BeagleBone Black shown at Figure 1.4 and detailed hardware specification at Table 1.2.

| | |
|---|---|
| **SoC**: | Broadcom BCM2837 |
| **CPU**: | AM335x 1GHz ARM® Cortex-A8 |
| **GPU**: | SGX530 3D |
| **RAM**: | 512MB DDR3 RAM |
| **Memory**: | 4GB 8-bit eMMC on-board flash storage |
| **Networking**: | 10/100 Ethernet |
| **Storage**: | microSD |
| **GPIO**: | 2x 46 pin headers |
| **Ports**: | HDMI, Ethernet, LCD, GPMC,MMC1/2, 4 Serial Ports, CAN0 |

**Table 1.2:** BeagleBone Black: Hardware specification

As we can see in comparison to Raspberry Pi, BeagleBone is a SBC more oriented for controlling low-level application. However BeagleBone lacks any USB ports for connecting any peripherals, so to extend BeagleBone functionality one must buy expansion boards.

Which is not necessarily bad, but they might not be available in common as classic USB peripherals.

#### 1.4.1.3 The Intel® Edison

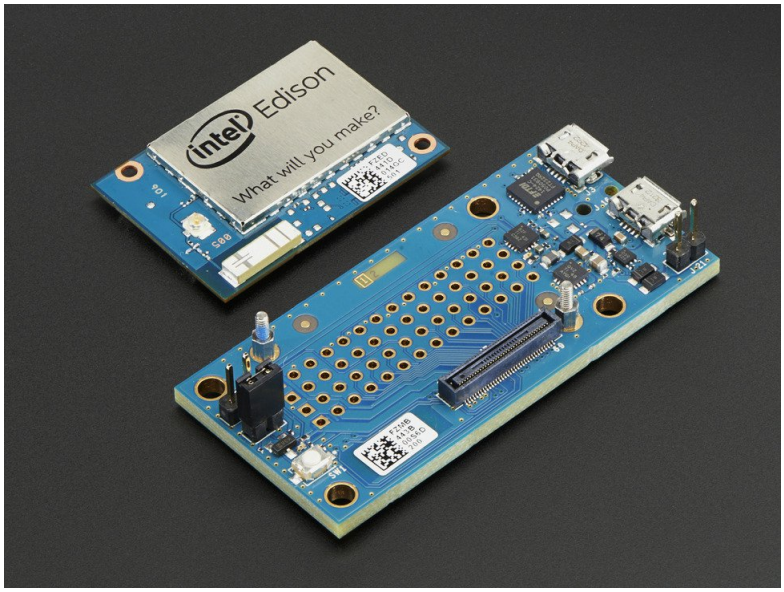



**Figure 1.5:** Intel® Edison with Breakout Board[4]

The Intel® Edison development platform is designed to lower the barriers to entry for a range of Inventors, Entrepreneurs and consumer product designers to rapidly prototype and produce IoT and wearable computing products.[12]
Intel® Edison is very small SBC which provides very interesting hardware specification, which can be found in Table 1.3. However to connect anything to Intel® Edison, expansion board is needed because I/O pins of Intel® Edison are grouped in very small 70 PIN I/O Connector. One example of expansion board is Edison Breakout board which has a minimalistic set of features and is slightly larger than the Edison module as shown at Figure 1.5. This is concept which is different from the Raspberry and the Beagles. If a project needs different set of capabilities instead of changing whole SBC, it is enough to change just the expansion board. One main disadvantage with starting with this platform is higher cost of obtaining Intel® Edison and exapnsion board.

| | |
|---|---|
| **SoC**: | 22-nm Intel® SoC |
| **CPU**: | dual-core, dual threaded Intel® AtomTM CPU at 500Mhz and a 32-bit Intel® QuarkTM microcontroller at 100 MHz |
| **GPU**: | SGX530 3D |
| **RAM**: | 1 GB LPDDR3 POP memory |
| **Memory**: | 4GB eMMC |
| **Wireless**: | Broadcom* 43340 802.11 a/b/g/n; Dual-band (2.4 and 5 GHz) On board antenna or external antenna |
| **Bluetooth**: | Bluetooth BT 4.0 |
| **Storage**: | microSD |
| **GPIO**: | 70 pin headers, populated |
| **Ports**: | USB OTG, 2 Serial Ports |

**Table 1.3:** Intel® Edison: Hardware specification

### 1.4.2 Peripherals

To provide necessary capabilities to solve this bachelor thesis, we might need peripheral devices to extend capabilities of developed embedded device.

#### 1.4.2.1 Bluetooth / WiFi Combination USB Dongle


**Figure 1.6:** Bluetooth / WiFi Combination USB Dongle[3]

If SBC chosen to complete this bachelor thesis would not have WiFi/Bluetooth capabilities, this adapter in Figure 1.6 will be used to provide it. Technical details can be found in Appendix B. Advantage of this USB dongle is that it 'saves' one USB slot by integrating WiFi and Bluetooth technology into one device.

#### 1.4.2.2 Adafruit 10-DOF

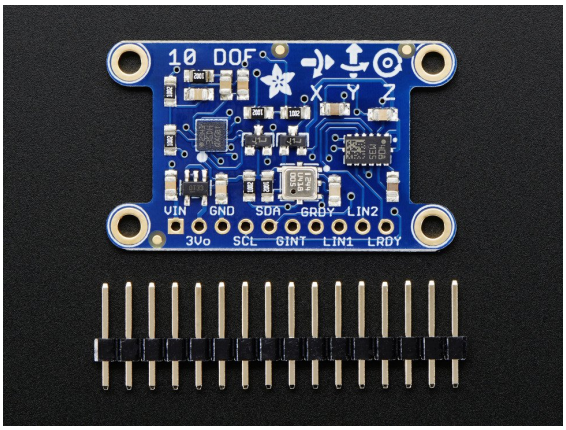



**Figure 1.7:** Adafruit 10-DOF IMU Breakout[44]

This inertial-measurement-unit(Figure 1.7) combines 3 sensors to provide 11 axes of data: 3 axes of accelerometer data, 3 axes gyroscopic, 3 axes magnetic (compass), barometric pressure/altitude and temperature. Since all of them use I2C, you can communicate with all of them using only two wires. On the breakout board are the data ready and interrupt pins available for advanced users.[44].

#### 1.4.2.3 Adafruit FONA 3G Cellular + GPS

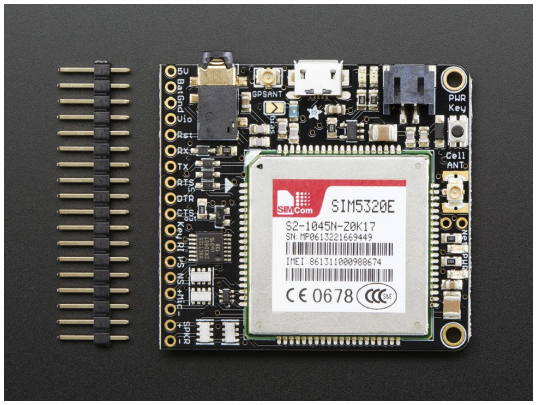



**Figure 1.8:** Adafruit FONA 3G Cellular Breakout - European version[2]

The FONA 3G has good coverage, GSM backwards-compatibility and even has a built-in GPS module for geolocation and asset tracking. This all-in-one cellular phone module that lets you add location-tracking, voice, text, SMS and data to your project in a single breakout as shown in Figure 1.8.[2]

## 1.5 Automotive electronic systems

As we will be gathering data from electronic systems of the car, lets overview how they work, what systems and protocols are present in modern cars and how to use them to our benefit.

### 1.5.1 Perspective

To gain better perspective into how complex electronic systems are in cars, lets go back a little into history. In 2002, Leen -Heffernan said in their work "Expanding automotive electronic systems":

> The growth of electronic systems has had implications for vehicle engineering. For example, today's high-end vehicles may have more than 4 kilometers of wiring-compared to 45 meters in vehicles manufactured in 1955. In July 1969, Apollo 11 employed a little more than 150 Kbytes of on-board memory to go to the moon and back. Just 30 years later, a family car might use 500 Kbytes to keep the CD player from skipping tracks. ([25])

It is now 2016 and electronic systems evolved so much that you can find 100 million lines of code in software of average modern high-end car[10]. Modern cars nowadays have more than 30-50 Electronic Control Units(Electronic Control Units (ECUs)). At Figure 1.9 we can see network of electronic components inside a car.

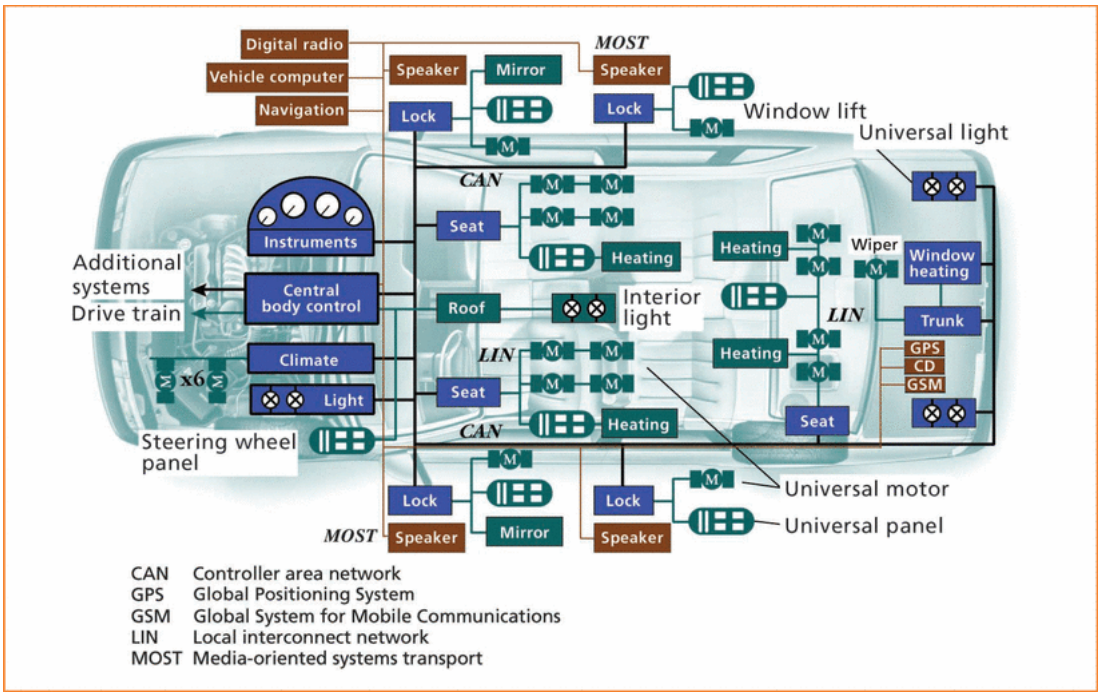



**Figure 1.9:** Vehicles electronic network [25].

#### 1.5.1.1 In-vehicle networks

Just as LAN or Wi-Fi connects computers, control networks connects a car's electronic equipment. By interconnecting different pars of ECUs, vehicle is able to share information amongst all distributed applications. In the history of automotive industry, wiring was just connecting element A to element B, however when electronic content in cars increased, the use of more and more cables to link elements grown enormously. To provide an example, Motorola reported in a 1998 press release, that replacing wiring cables with LANs in the four doors of a BMW car reduced the weight by 15 kilograms.[25] Point-to-point wiring was replaced first by centralized then distributed networks in the early 1980s.[26]

### 1.5.2 Controller Area Network

The Controller Area Network (CAN) is an ISO defined serial communications bus specifically designed with real-time requirements in mind. Developed in the 1980s by Robert Bosch, its ease of use and low cost has led to its wide adoption throughout the automotive and automation industries[15]. It is currently the most widely used vehicular network[25]. CAN is designed for signaling rates up to 1 Mbps, high protection against electrical interference, and an ability to find and repair data errors.

The CAN communications protocols, ISO 11898, conform to the Open Systems Interconnection (OSI) model that is defined in terms of layers[11]. Protocols specify how data travel between devices on the network. Physical layer defines how to communicate between devices which are connected. Lowest two layers of the ISO/OSI model are defined as the data-link and physical layer shown in Figure 1.10.

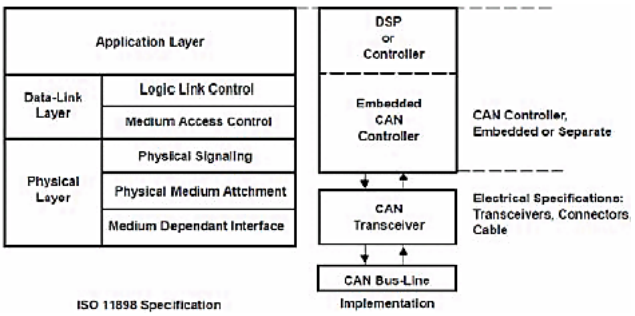



**Figure 1.10:** Layers of CAN Communications protocol, ISO 11898[11].

When data is transmitted over a CAN network no individual devices are addressed. Instead, all messages have an unique identifier, a unique tag of its data content. Not only it identifies the message contents, but also the message priority as how much important data contents are. Lower the message priority, the more important is the message and it is delivered in prior to messages with higher priority. When a device connected to the network wants to transmit

message, it passes the data and the identifier to the CAN controller with a relevant request to transmit information. CAN controller formats the message contents and transmits the data in form of a CAN frame as shown in Figure 1.11.

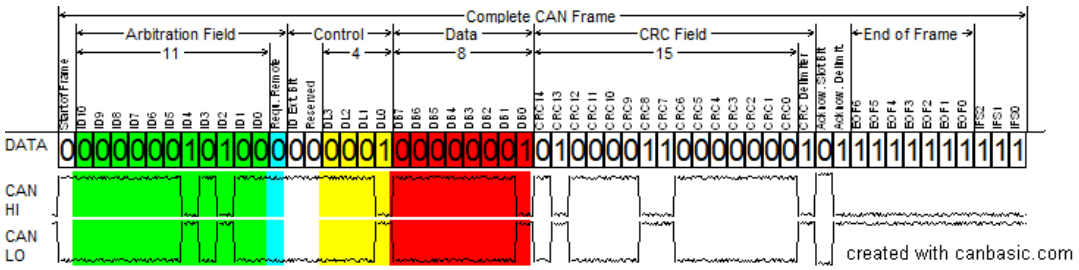


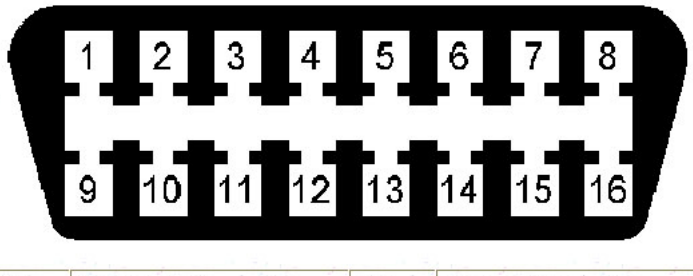

**Figure 1.11:** CAN bus frame [14]

### 1.5.3 Conclusion

By understanding standard messaging protocol used in cars, we are able to read and create messages. Creating and reading messages relies on correct construction of data payload and assignment of correct identifier to the message, however specification of how to create this particular parts of the message is not publicly available, moreover specification is proprietary and probably different for every car manufacturer. To discover what data in messages are and what identifier are used, data analysis of closed data dumps from different states of the car(Ignition on, windows down, engine off, etc.) can be used, however this method is dependent on concrete make and model of the car, and is not suitable for large scale applications or general use.

## 1.6 OBD-II

When environmental pollution became a world issue in the 1970s, the U.S. created the Environmental Protection Agency (Environmental Protection Agency (EPA)). EPA established a new standard to limit the environmental pollutants emitted from vehicles. The car makers devised an electronic control system for the fuel supply and ignition devices, based on the standard. In addition, the Society of Automotive Engineers (Society of Automotive Engineers (SAE)) established OBD in 1988 as a standard for the plug connector and on-board diagnosis system[22] as shown in Figure 1.12.




| PIN | DESCRIPTION | PIN | DESCRIPTION |
|---|---|---|---|
| 1 | Vendor Option | 9 | Vendor Option |
| 2 | J1850 Bus + | 10 | j1850 BUS |
| 3 | Vendor Option | 11 | Vendor Option |
| 4 | Chassis Ground | 12 | Vendor Option |
| 5 | Signal Ground | 13 | Vendor Option |
| 6 | CAN (J-2234) High | 14 | CAN (J-2234) Low |
| 7 | ISO 9141-2 K-Line | 15 | ISO 9141-2 Low |
| 8 | Vendor Option | 16 | Battery Power |

**OBD-II Connector and Pinout**

**Figure 1.12:** Pin-out of the plug connector

OBD standard developed over-time and in time of writing this thesis OBD-II is standard which is used in the industry.

### 1.6.1 Standard

The OBD-II standard specifies the type of the connector and its pin-out, the messaging format and electrical signaling protocols available. Difficulties may arise because of the deficiency in the OEM specific implementation of the standard. According to The OBD Home Page: There are five basic OBD protocols in use, each with minor variations on the communication pattern between the on-board diagnostic computer and the scanner console or tool. Formula CAN is the newest protocol added to the OBD specification, and it is mandated for all 2008 and newer model years[41]. Original equipment manufacturer (OEM) provides a list of vehicle queries which can be send to the OBD-II port for diagnostics and gathering real-time parameters, but the list may vary from car to car. Each of this queries has a specified response on how to decode returned data again by OEM and may vary from another manufacturer responses. Finally, the OBD-II standard provides an extensible list of DTCs(Diagnostic trouble codes). As a result of this standardization, a single device can query the on-board computer(s) in any vehicle [40], however this is again based solely on OEM so for getting all possible informations from the OBD-II port, specification from OEM is needed.

### 1.6.2 Communication

As mentioned above to access information from sensors attached to a vehicle via a OBD-II port, special codes called Parameter IDs(Parameter IDs (PIDs)) are used. Typically a process of retrieving data from vehicle trough OBD-II port is as follows.

1. Requested PID is entered either through connected computer or scan tool.
2. Request is send to vehicle CAN bus.
3. Device which is responsible for given PID recognizes it and then reports the value back to the bus
4. Connected computer or scan tool reads the response.

**Figure 1.13:** Query OBD-II process

From Figure 1.13 we can see that OBD-II port is in simple terms subset of CAN frames. Identifier and data payload is supplemented from OBD-II protocol based on concrete PID query.

### 1.6.3 ELM-USB

ELM-USB is OBD-II to USB interface compatible with ELM-32x series (ELM323, ELM327, ELM CAN).[36] It supports any of the five communication protocols used in OBD-II compliant vehicles shown in Figure 1.14.

- SAE J1850 PWM
- SAE J1850 VPW
- ISO9141-2
- ISO14230-4 (KWP2000)
- SO 15765-4/SAE J2480

**Figure 1.14:** OBD-II compliant vehicles protocols

#### 1.6.3.1 Connection

As ELM-USB is very easily connected, the OBD-II connector needs to be connected into OBD-II port of the vehicle and USB connector to USB port of the embedded device. After connecting, operating system recognizes new device and instantiate a serial port for sending

commands to the ELM-USB device. Commands are in form of AT commands which are similar to the AT commands of the Adafruit 3G FONA used as a modem for cellular network.

### 1.6.4 Conclusion

Main advantage of OBD-II port is that obtaining a single parameter from the car is relatively simple. It is enough to know correct PID to query to send, but it is not suitable for real-time telemetry, because of repetition rate of the available data. The more different data is read out, the lower repetition time will be for each queried type, due to the fact that information is only accessible by question answer basis.

## 1.7 Node.js

Node.js is a event-based network application which allows developers to use asynchronous programming interfaces for I/O operations. Language which is primarily used with Node.js platform is JavaScript. Despite it is one of the newer platforms available, Node.js has significant community of developers and we can see more and more applications build in the industry on top of the Node.js. Node.js is built on a V8 JavaScript Engine.

### 1.7.1 V8 JavaScript Engine

V8 JavaScript Engine is an open source engine developed by Google for the Chrome web browser. V8 JavaScript Engine (V8) is open source high-performance JavaScript engine, written in C++. V8 implements ECMAScript as specified in ECMA-262, 5th edition, and runs on Windows (XP or newer), Mac OS X (10.5 or newer), and Linux systems that use IA-32, x64, or ARM processors. In several benchmark tests, V8 is many times faster than JScript (in Internet Explorer), SpiderMonkey (in Firefox), and JavaScriptCore (in Safari).[17]

### 1.7.2 Event Loop Mechanism

Every Node.js application runs a single-threaded continuous event loop, which sources out time-consuming tasks to worker threads, which callback the main event loop when tasks is finished, as shown in Figure 1.15.

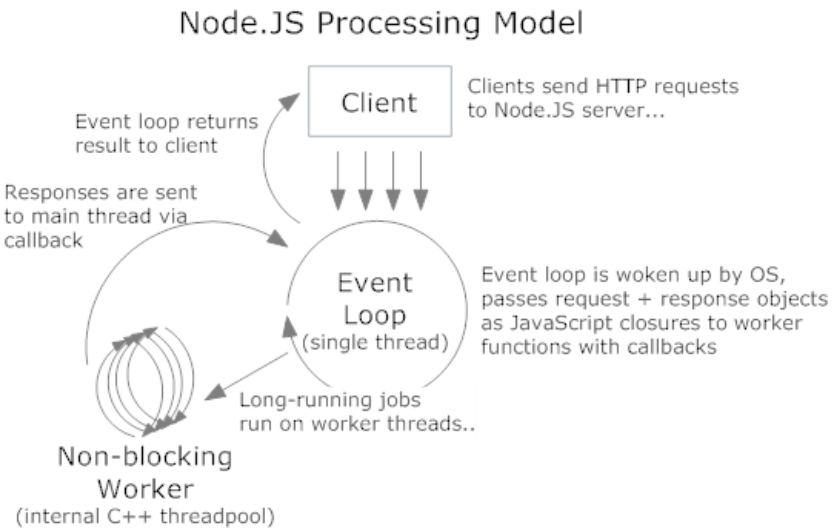



**Figure 1.15:** Node.js Process Model[45]

Event-driven programming offers more scalable alternative than multi-threaded programming, that provide developers with much more control over application behavior and activities. In this model, the application firstly registers interest of certain events, such as disconnection of the client, data ready to be read, end of stream, etc. Then application waits to be notified of occurrence in these registered events, and can act upon triggering. [18] When an event happens a record of the event is added to the Node.js event queue, after main thread looks at the event queue and takes the first event from the front and starts running the corresponding event handler. After event handler finishes the thread control goes back to the main thread which again looks at the event queue if there are any events left to process.[42]

### 1.7.3 Asynchronous Model

Now we know that single threaded event system works by placing events in a queue and process them one after another by calling appropriate event handler. The event handler then runs until all applications registered for that event are completed and returns control to the dispatcher, which looks again at the event queue for another event. This is an asynchronous system because order of execution of application code is not guaranteed. The order is based on event triggering which can not be pre-determined. To put it another way, order of the application written by event-driven model might be any of the combination of events which that concrete application react to. However, problem might be of the single threading, as Node.js is.

#### 1.7.3.1 Problem

The reason is simple, what if event handler takes too much time to finish, or the worst case scenario never returns thread control to the dispatcher – application seems to freeze. Event handler never ends, so no other event handler from event queue ever gets to run.

#### 1.7.3.2 Solution

Solution is simply use event handler to set up new thread and get it to do all the work. After that event handler returns at once and the new thread can process time consuming work. Problems of synchronization arise when creating a new thread in single-threaded application. In Node.js time-consuming functions are dispatched to non-blocking workers as shown in Figure 1.15. Synchronization problem is we do not know when worker finishes with particular function, and usually we would like to process return value of that function somewhere else in the code. This problem might be solved in many ways, but all of them are nearly variations on the idea of passing an explicit something to do when function has finished - i.e callbacks. Callback is function which will be executed at the end of function dispatched to the worker, which successfully solves problem of how and when to process return value of unknown time running function.[21]

# 2. Specification

## 2.1 Requirements

In this chapter we provide clear requirements for this thesis solution. By stating clear specifiaction the result of this thesis can be evaluated by comapring results and requirements.

### 2.1.1 Hardware

To complete hardware requirements an embedded device must be developed. Operating system of this embedded device should be the most suitable for the device. Peripheral devices needed to be connected to this device are:

- Adafruit 10-DOF to enable device to gather accelerometer data through $I^2C$ bus
- Adafruit FONA 3G Cellular Breakout for connecting to the Internet through cellular network and ability to send SMS messages
- Wi-Fi extension if device does not have wireless connection built-in, to provide means of connecting to the device without need of serial port

After peripheral devices are connected, operating system needs to be configured that Wireless access point is created automaticaly after device start-up. Connection between OBD-II port and the device have to be established. Solution must be provided how to configure and gather data from OBD-II port. Every peripheral device needs to be configured properly to enable operating system communication with each of them.

### 2.1.2 Application

Application have to be developed for collecting information from different sources. Application to be more easily configurable, develop modular system which objectifies every data gathering application in form of interchangeable modules. These modules have to be able accept and send data information. Interface or template should be provided for easy creation of new modules. Modules must be selectable by type, to provide manner of choosing

same type of modules by unique type name. Routing logic should be developed to manage flow of information to and from modules. Defining routes must be provided by key value pattern where key can be either unique name or type of the module/s. Key in routes is to provide source module from which data are being sent. Values are structured by array and are also being denoted by unique name or type of the module. Values of the routes are defined as modules which can accept data. Interface needs to be developed for sending and accepting data to provide unified access to the objectified modules of your application. Lower-level application modules should be produced for:

- Gathering sensoric data
- Gathering all OBD-II parameters supported by car
- Sending SMS messages
- Sending data over to the server on the Internet by AMQP protocol
- Sending trigger events to the IFTTT webservice
- Storing data to the Redis database
- Gathering NMEA data from GPS sensor
- Gathering time from Operating System
- Sending data to the standard output of the Node.js application

By utilizing this lower-level application, higher-level application modules must be developed for:

- Reporting accidents over SMS messages, by providing info about driver, location of the accident and maximal g-force reading recored by the accelerometer
- Advising to shift gears by reading RPM information from OBD-II port by sending trigger event to the IFTTT webservice.
- Bulk data sending over the cellular network by gathering all created data and sending them one in a time, not as a constant stream of information.

Routes shall be provided correctly for application to work by interconnecting lower-level modules to enable higher-level modules to work. Configuration of the modules should be provided by creating web application developed for changing configuration of any module or route. Web application needs to be able to display last created data from each module, also web application should be accessible by connecting to the device's wireless access point.

### 2.1.3 Output

Output of this bachelor thesis should be an application which upon correct instalation on embedded device is capable of monitoring various inputs of sensoric data, based upon configuration distribute and make them accessible for different services.

# 3. Solution

In this chapter we provide a solution for this bachelor thesis problem.

## 3.1 Hardware

To begin with we need device which will be capable of running operating system. There are many various platforms which provide single board computers, we have chosen raspberry platform as most suitable because it is capable of running full Linux operating system, have 4 USB slots which will be needed for peripheral devices and foremost there is a plethora of hardware add-ons, sensors and boards which works with raspberry models, not to mention availability of information how to connect them.

### 3.1.1 Device

Since we will be building embedded device which will allow us to gather data and send them over to the Internet, it is necessary to choose a single-board computer which provides means for connecting to the Internet either by connecting using built-in hardware or peripheral device. For connecting various sensors SBC must provide enough General-purpose input/output (GPIO) pins and support communication protocols commonly used with sensors e.g $I^2C$. After picking the right platform we need to pick model. Newest model on the market from raspberry platform in time of developing this thesis was Raspberry Pi Zero which was not suitable for the needs of this thesis because it does not have any usb ports. It has support for On-the-go USB cables but we need device which supports at least 2 or 3 usb ports, which we would need to solve by buying USB hubs. We will use Raspberry Pi 2 Model B, as we own one. It has 4 USB ports, 40 GPIO pins and micro SD card slot. This device covers almost all the needs of this thesis. We will need to extend it with WiFi/Bluetooth USB dongle which will enable us to create WiFi access point. In time of writing this thesis Raspberry Foundation released new device Raspberry Pi 3 which is even more suitable for us because it has built-in Wireless LAN so no WiFi/Bluetooth USB dongle is necessary.

### 3.1.2 Operating System

The most suitable operating system for our needs is Linux based. Concrete distribution of Linux based operating system will be Arch Linux. The main gain of using Arch Linux is it is extremely lightweight distribution with Keep it Simple and Stupid(KISS) principle in mind also it is a rolling release which means it is always up-to-date. Port of Arch Linux distribution is Arch Linux ARM which carries forward the Arch Linux philosophy of simplicity and user-centrism, targeting and accommodating competent Linux users by giving them complete control and responsibility over the system. Instructions are provided to assist in navigating the nuances of installation on the various ARM platforms; however, the system itself will offer little assistance to the user. Installing and basic configuration of Arch Linux will not be covered in this thesis, as there is enough information available to achieve it. Arch Linux has one of the best documentation online, for those who are not familiar with how to install and configure this particular Linux distribution, follow documentation from Arch Linux website.

## 3.2 Configuration

In this section we will cover configuration of various applications and system properties to set up the device correctly for our needs.

### 3.2.1 Database

For storing information on provided device, it needs to be capable of running database. As it is with Linux, there are a lot of databases to choose from and to suit our needs we need to be aware on one fact, and that is where all gathered information will be stored. Raspberry can support very large memory cards, up to 512 gigabytes, and with various read/write speeds, so one could find card which will suit oneself needs.[34] In spite of availability of different micro SD cards, every card has limited read/write cycles after which it becomes unreliable. Another solution is not to store information on card at all. Typical Linux database will create database file to which it stores data, in configuration we could point database to create this file in a special directory. On Linux it is called /tmp[31], this directory is allocated not on the memory card but instead on RAM memory, which would save us need of large and expensive card. It comes with disadvantage, after restart or shutdown of the device, we would not be able to access stored data as RAM memory is volatile[43]. We will use Redis database which has the ability to set the database in RAM memory but it is configurable to save snapshots(copies) to the micro SD card. Redis is an open source, in-memory data structure store, used as database, cache and message broker[33]. Configuration file for database which runs on our device can be found in resources of this bachelor thesis under name `redis.conf`.

### 3.2.2 Wireless Access Point

To automatically create wireless access point when the device boots up we need several configurations and applications. To create software access point it is necessary to have nl80211 compatible wireless device, which support AP operating mode[39]. Application which will be needed is `hostapd`, `iw` and `dhcpcd`. Configuration file for hostapd can be found in resources under name `hostapd.conf` which needs to be copied into `/etc/hostapd/`, and a systemctl rule `wifi-ap.service`, which needs to be copied into `/lib/systemd/system/`. After that systemctl daemon have to be reloaded, then start and enable wifi-ap.service. If you have compliant device it will automatically create wireless Access Point which is defined in hostapd.conf.

### 3.2.3 Internet

As we are using Adafruit FONA 3G Cellular + GPS modem, creating connection to the Internet is possible in different ways. First is using pppd daemon which is capable upon configuration to create ppp0 device which will be used to connect to the Internet. Configuration files for ppp can be found in resources in `ppp` folder. Another approach is with wvdial, which configuration can also be found in repository under name `wvdial.conf`. At last FONA 3G is supported by QMI modem protocol. For dialing with qmi it is needed to have installed `libqmi` and `net-tools` packages. Name of the apn server to which modem needs to connect from your mobile Internet service provider is required. Replace that APN server in `qmi-network.conf`, which is located in the resources and copy it to `/etc/`. From script folder found in thesis repository copy a helper script, and then simple `qmi_setup.sh start` starts the connection to the Internet. If SIM card is PIN locked it is necessary to unlock it manually trough serial console or with help from `mmcli` command.

## 3.3 Peripherals

For providing proof of concept device which is capable of gathering information from various sources we will connect to the device numerous peripherals which will generate information. Peripherals will be described in upcoming subsections and configuration instructions will be provided. We believe that connecting this peripherals is such trivial that it does not need to be explained to the reader, if however someone would not have sufficient knowledge of how to connect these to the raspberry pi we suggest to find some guide on the Internet for correct connections to be made. Overview of peripherals connected to device is provided below in Table 3.1.

| Peripheral device | Connected by |
|---|---|
| Wireless/Bluetooth dongle | USB |
| Adafruit 10-DOF | $I^2C$ bus |
| Adafruit FONA 3G Cellular + GPS | USB(creates 4 virtual serial ports) |
| ELM327 Compatible device | USB |

**Table 3.1:** Peripheral devices connected to Raspberry Pi

### 3.3.1 Adafruit 10-DOF

To connect this board device will need to have installed packages `i2c-tools` and `lm_sensors`. After copying `config.txt` from thesis resources to `/boot/config.txt`, enable modules `i2c-dev, i2c-bcm2708` in `/etc/modules-load.d/raspberrypi.conf`. That is it, now with command `i2cdetect -y 0` it should work, if not follow troubleshoot documentation on raspberry pi on archlinux wiki[32].

### 3.3.2 Adafruit FONA 3G Cellular + GPS

There are multiple ways how to connect to the Internet, send SMS and make voice calls, all of them boils down to using right AT commands through serial console which is automatically created by the kernel. Connecting modem through usb to the device, kernel will create 4 serial ports(debug,nmea,serial,modem). After starting gps chip with correct command `at+cgps=1` in serial port GPS chip will start to push nmea sentences on nmea port at 115200 baud rate.[38]

### 3.3.3 OBD-II connection

Connection to OBD-II port is possible by various devices which interface OBD-II to other standards such as USB, $I^2C$, SPI, CAN. To be able to connect to the OBD-II port by various devices not only by Raspberry Pi, we have chosen ELM-USB.

#### 3.3.3.1 Configuration

To be able successfully connect to the OBD-II port of the vehicle it is necessary to select proper protocol to communicate. There are two possibilities how to do that. ELM-USB device can be configured to use Automatic protocol detection or specifically instructed to use defined protocol. Command for this configuration is `AT SP X`[37] where x is one of values defined at Table 3.2

| X | Communication protocol |
|---|---|
| 0 | Automatic protocol detection |
| 1 | SAE J1850 PWM (41.6 kbaud) |
| 2 | SAE J1850 VPW (10.4 kbaud) |
| 3 | ISO 9141-2 (5 baud init, 10.4 kbaud) |
| 4 | ISO 14230-4 KWP (5 baud init, 10.4 kbaud) |
| 5 | ISO 14230-4 KWP (fast init, 10.4 kbaud) |
| 6 | ISO 15765-4 CAN (11 bit ID, 500 kbaud) |
| 7 | ISO 15765-4 CAN (29 bit ID, 500 kbaud) |
| 8 | ISO 15765-4 CAN (11 bit ID, 250 kbaud) - used mainly on utility vehicles and Volvo |
| 9 | ISO 15765-4 CAN (29 bit ID, 250 kbaud) - used mainly on utility vehicles and Volvo |

**Table 3.2:** Communication protocols of ELM-USB

## 3.4 Software

### 3.4.1 Problem abstraction

To solve given task, lets look at data gathering and what can be done to optimize process of collecting information from often very different sources. For correct conclusion to be made, we need to abstract the meaning of car connection as merely only as one of information sources. To solve how to easily interconnect different input sources(information producing) to even more different output sinks(information sending), a framework was developed. A data application module which will provide easy connecting of different sources with data

producing modules to any data accepting modules. So to begin lets define some construction blocks of this framework so we can easily build a suitable application.

### 3.4.2 Data-Logger framework

Data-Logger framework is a stand alone application module which can be incorporated in any Node.js application. Main principle of the framework can be described by pipes and filters design pattern. In pipes and filters design pattern a pipeline consists of chain of processing elements arranged in a way where output of each element is the input of the next, same as physical pipes together to form a pipeline. In the framework the information which is passed along is data collected by source modules from devices which generate data e.g. sensors. Pipes and filters design can be viewed as form of functional programming, using streams as data objects. They could even be seen as a particular form of monad for I/O[29]. Main goal of this framework is to provide easy way how to setup communication between various inputs(sensoric data, IoT devices, cars, etc.) and outputs(database, application, server, cloud, etc). Framework is structured by main app, modules and routes. Their relations are as shown in Figure 3.1.

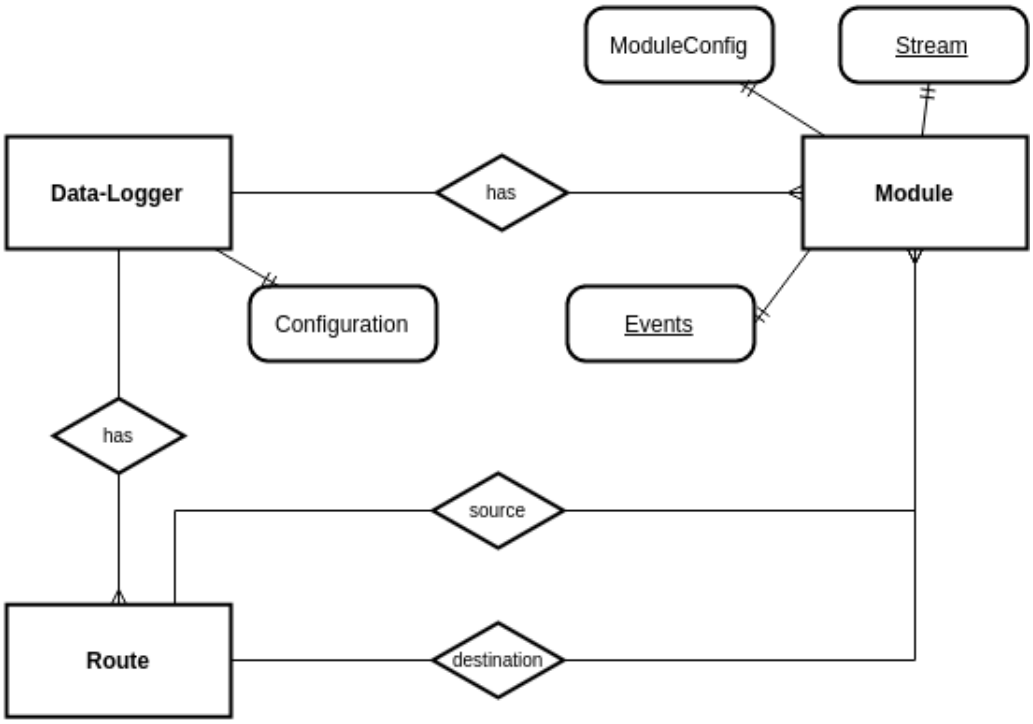



**Figure 3.1:** Entity diagram of Data-Logger framework

Modules are stand alone application modules which are written to obtain,modify, or send data to various devices or services. Pipes and filters design is applied when connecting modules together. How are modules connected is defined by routes. which are an interface

for configuring which module will be streaming data to which. Data-Logger is configurable through internal function and by file which serves for storing configurations of each model and route. To help understand dataflow of the Data-Logger we provide diagram in Figure 3.2.

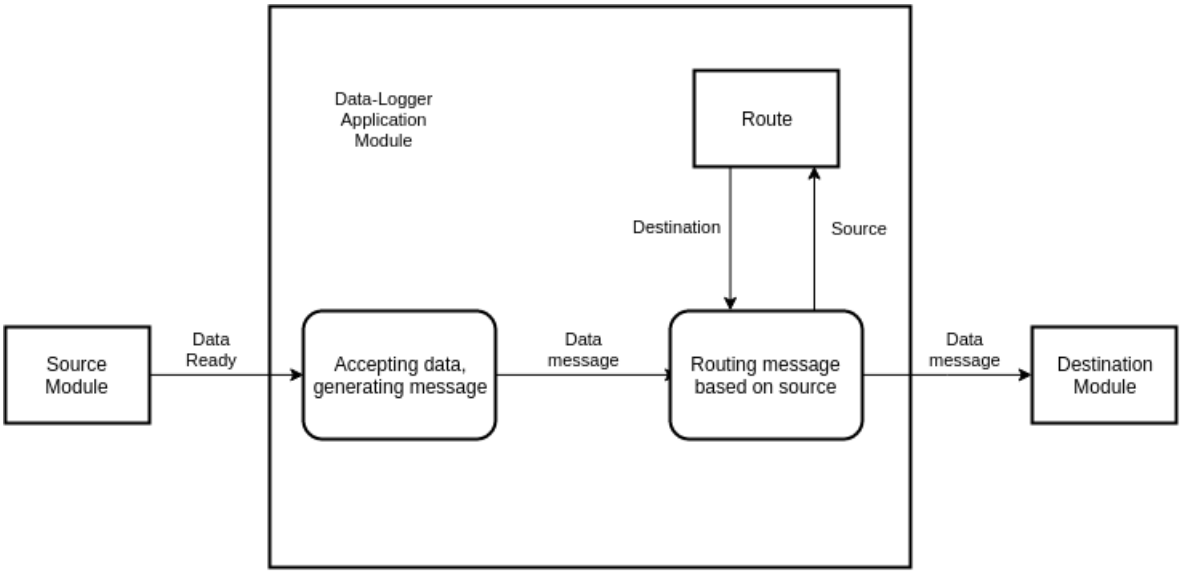



**Figure 3.2:** Dataflow of Data-Logger framework

#### 3.4.2.1 Module

Application Module is basic structure for every element in Data-Logger. It is an encapsulating class for every information creating or sending application. By this it is possible for framework to objectify every application, no matter how much they would be different, and extract data from generating one or send data to to accepting one. With this feature we will be able to get data from OBD port of the car system the same way as from accelerometer present at Adafruit 10 DOF sensoric board. This is achieved by writing specific applications to parse/get data from the source, which are then run under Data-Logger framework. Every module can be generating data, or accepting data and acting upon it. For module to be generating or accepting one, module must be implemented by the same interface so it can be easily handled by Data-Logger, otherwise for every application connected to Data-Logger there would had to be specific functions to run to obtain or accept data. Configuring modules are easy as from framework we can pass settings for every module separately, either from application or much easier from a main configuration file. This configuration file has a specific block of configuration settings for every module present in current implementation of Data-Logger. To provide better description of what data-logger module consists we provide activity diagram in Figure 3.3

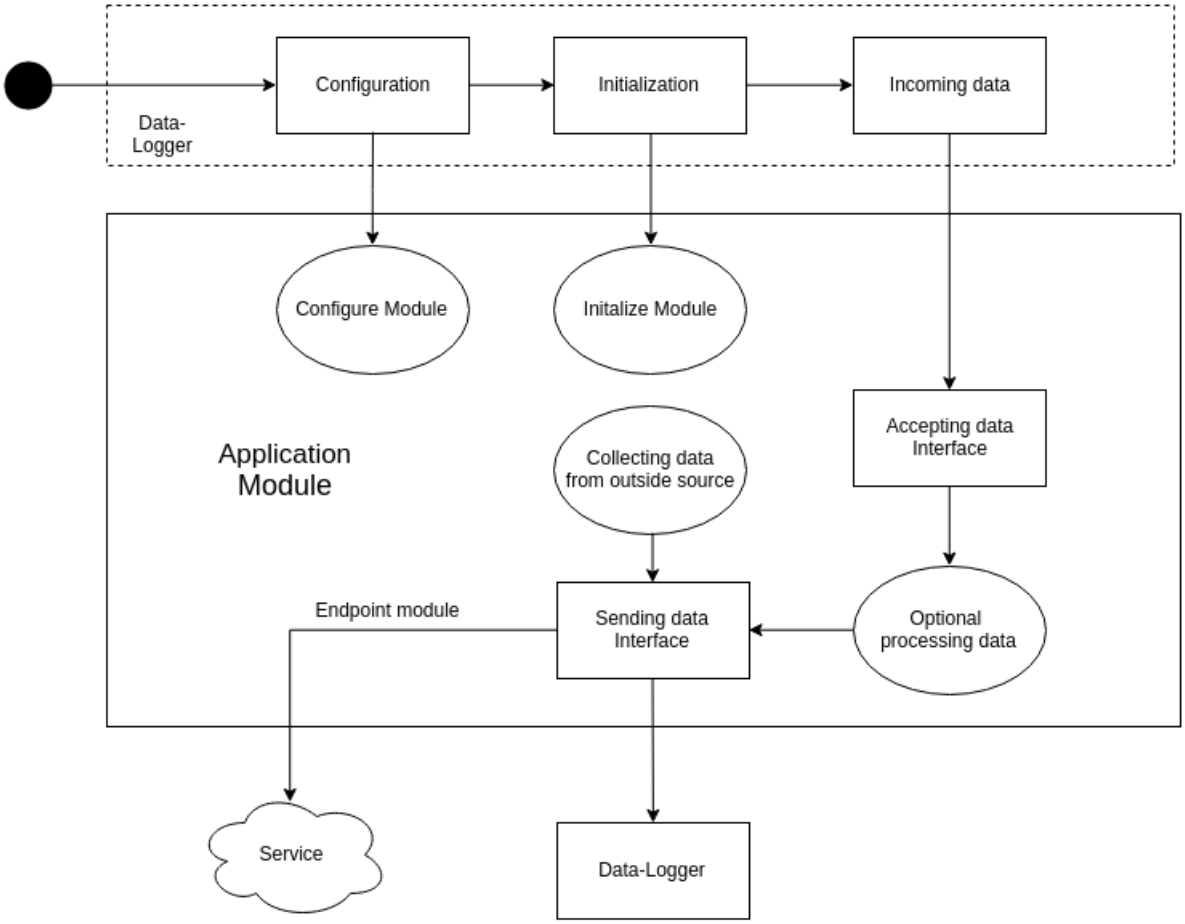



**Figure 3.3:** Activity of Data-Logger module

#### 3.4.2.2 Route

Route is another basic structure how to define a dataflow from or to every module. Routes are defined with simple principle in mind: you define source and then every possible sink to which data have to be sent. By this configuration Data-Logger will start sending or receiving data from modules depending on which side of the equation they are. If the module is generating data, framework will use specific general interface to gather data provided by the module and then distribute it to every sink defined by the route again by using general interface to sending data to module. With this principle one can define a very detailed dataflow. If we would scale this framework, say we would have 10 sensors producing same data but they would need different modules to be handled, it would become soon tedious to write routes based only on unique identification matching of modules. So for every module there is a specific setting to be set, a type, by which we can more easily define routes, e.g. defined route as `accelerometer -> database` and from now on, every data produced by module which has type set to accelerometer will be routed to every module which has database set as type. Activity diagram of Data-Logger routes is shown in Figure 3.4.

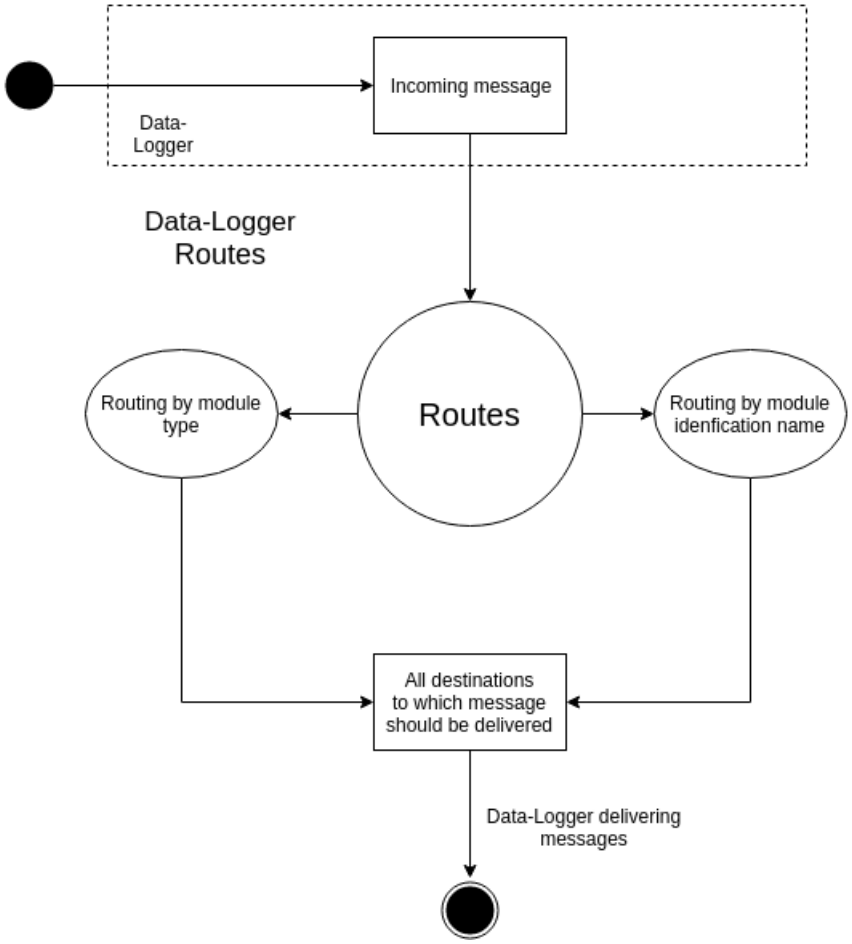



**Figure 3.4:** Activity of Data-Logger routes

### 3.4.3 Data-Logger low-level modules

In this section of bachelor thesis we will describe modules which were developed in order to help complete the assignment. Low-level modules are modules which does not implement any logic or process data in any way. They are developed for simple reason of data gathering or sending. If they receive messages, low-level module does not alter contents of messages, just sends them to the destination determined by the module.

#### 3.4.3.1 Blank module

During developing modules for this bachelor thesis, a blank module was created which has all the necessary functions, interface to make more easier creation of new modules. Code is commented to provide viable information what is expected from different functions in case their name would not be clarifying enough. Blank module combines source and sink functionality together but they do not need to be used both.

#### 3.4.3.2 Time module

This is rather simple module which will be used to beta test others modules. Advantage of having this module is not being dependent on any hardware. In periodic intervals defined by sample rate, current time as data is provided.

#### 3.4.3.3 Accelerometer module

This module is dependent on Adafruit 10 DOF sensor board and will utilize on-board LSM303 accelerometer sensor, communicating over I2C protocol. Module will emit proper acceleration in 3 axis, maximum acceleration +- 16g in each direction, sample rate will be configurable. LSM303 accelerometer sensor supports various settings as which axis acceleration is registered, how fast are these values checked, the constants for these settings are implemented, but any benefit was not found of selecting only one axis or sampling at lower rate, so defaults are every axis is registered, and sampled as fast as it can be for accident checking module.

#### 3.4.3.4 GPS module

GPS module as name suggests will parse nmea sentences from FONA 3g modem which has built-in GPS chip. AS for this module there is not much to be configured, because modem emits nmea sentences at 115200 baud rate as fast as it can, meaning every second, only thing found which could be configured is to switch output port for nmea sentences from nmea to serial, but then sending SMS and calling would not be possible, so it is not a feasible setting.

#### 3.4.3.5 SMS module

This module is a sms message sending module, but it is not designed for general use. Message relays on incoming data to be structured correctly from accident module and send 3 SMS messages with information about driver, GPS position and maximum g-force registered by the accelerometer. Module requires modem to be pin unlocked otherwise SMS sending will fail. Sending SMS message with `AT+CGMS` command requires string to be sent with a maximal size of 150 characters and ended with `^Z` special character. After message is sent, module responds with OK and number of total SMS messages sent by the module.

#### 3.4.3.6 OBD-II module

By far the most challenging module in this thesis. Module connects with help of ELM327 compatible device to the OBD-II port and selects automatic protocol. Problem with automatic protocol selection is protocols have different initializing response, the module needed to be adapted to this discovery, to be working with older and newer cars, we successfully tested this module on 5 different cars in make and model, but we can not rule out the possibility of OBD module not working with specific model and make of car, because there are multiple protocols present and their specification is not publicly available for free.

After successful initialization, OBD-II module will poll the connected car for supported commands which creates a polling loop out of commands which are supported for that particular vehicle and periodically queries the data. One must be vary of this one limitation, and that is older cars have older protocols which have slow rate of sending data, and therefore sample rate of this module should be considered to be calculated correctly, otherwise buffer overflow might occur. Module implements simple buffer overflow logic, when length of PID commands queue becomes more than threshold OBD module will not add more commands to queue until the length lowers than threshold. Module emits data for every queried command separately as in some cases gathering information from every supported command would simply took too long time to collect.

#### 3.4.3.7 RabbitMQ module

RabbitMQ module is primary module for getting data from Raspberry to server or cloud service. Module client connects to rabbitMQ server, after which it creates queues on the server based on message header id. Implication is that on server we will have queues for every module uploading data. Server then can process this queues separately with data processing applications and have various services to evaluate data and/or act upon it. In testing this module average speed of delivering messages was 500 messages in one second, this rate is based on how big information is being passed onto the server, so for example data from bulk module might have less throughput than accelerometer data. Connection settings can be altered in configuration file.

#### 3.4.3.8 Redis module

Redis module is database module for storing data in Redis database. Redis stores data in mapping keys to values. For correct work module implements counters for every data source which wants to store data in database. Module stores whole given messages in hash keys. Key is composed from message header id and counter which is incremented by every message stored. As not to store counters in modules, counter current value is stored by message header id key. By configuring database snapshotting, Redis database is able to recreate data structure from previous sessions, and continue where application left, with correct counters so data would not be accidentally overridden. Database is password protected which is set by configuration file.

#### 3.4.3.9 IFTTT module

IFTTT module is a support module for communicating with web service called Maker. With this service one can create different trigger events for data which arrives to the event and connect them to various other web services as GMail, Facebook, Instagram, etc. Data format is limited as Maker channel supports JSON format with only 3 keys, namely `value1,value2,vaule3`, so complicated data structures are not suited for this

module. Values of this 3 keys can be any string, with this in mind, the limitation could be circumvented for example with CSV format of data, which can be automatically parsed by for example Google drive spreadsheet application. Fair to say is that this service is free of charge and provides with possibilities to connect data generated with vast number of popular web services.

#### 3.4.3.10 Console module

Very simple module which provides structured module for console logging data to show what information is passed into module. This module was primarily used for developing purposes same as time module. This module will probably not be used in production environment but is very handy for debugging purposes, when modules do not cooperate in expected way. Console header can be set only to show configuration capabilities in early stages of development.

### 3.4.4 Data-Logger high-level modules

High-level modules are modules that implement logic or process data. They are developed for data manipulation, grouping or analyzing, mainly by connecting low-level modules to them and analyzing data from different sources.

#### 3.4.4.1 RPM advisor module

RPM module is a module which listens for data from OBD-II module and evaluates only RPM(Revolutions per minute) data from car. Purpose of this module is to simulate modern cars which often provides gearshift indicator together with information when to shift. In configuration file can be find two values, downShiftTreshold and upShiftTreshold. When RPMs drops under downShiftTreshold threshold, module will generate data to indicate to shift gears, similar action happens with upShiftTreshold, but instead dropping RPMs, module triggers when revolutions are higher then threshold as can be seen in Figure 3.5

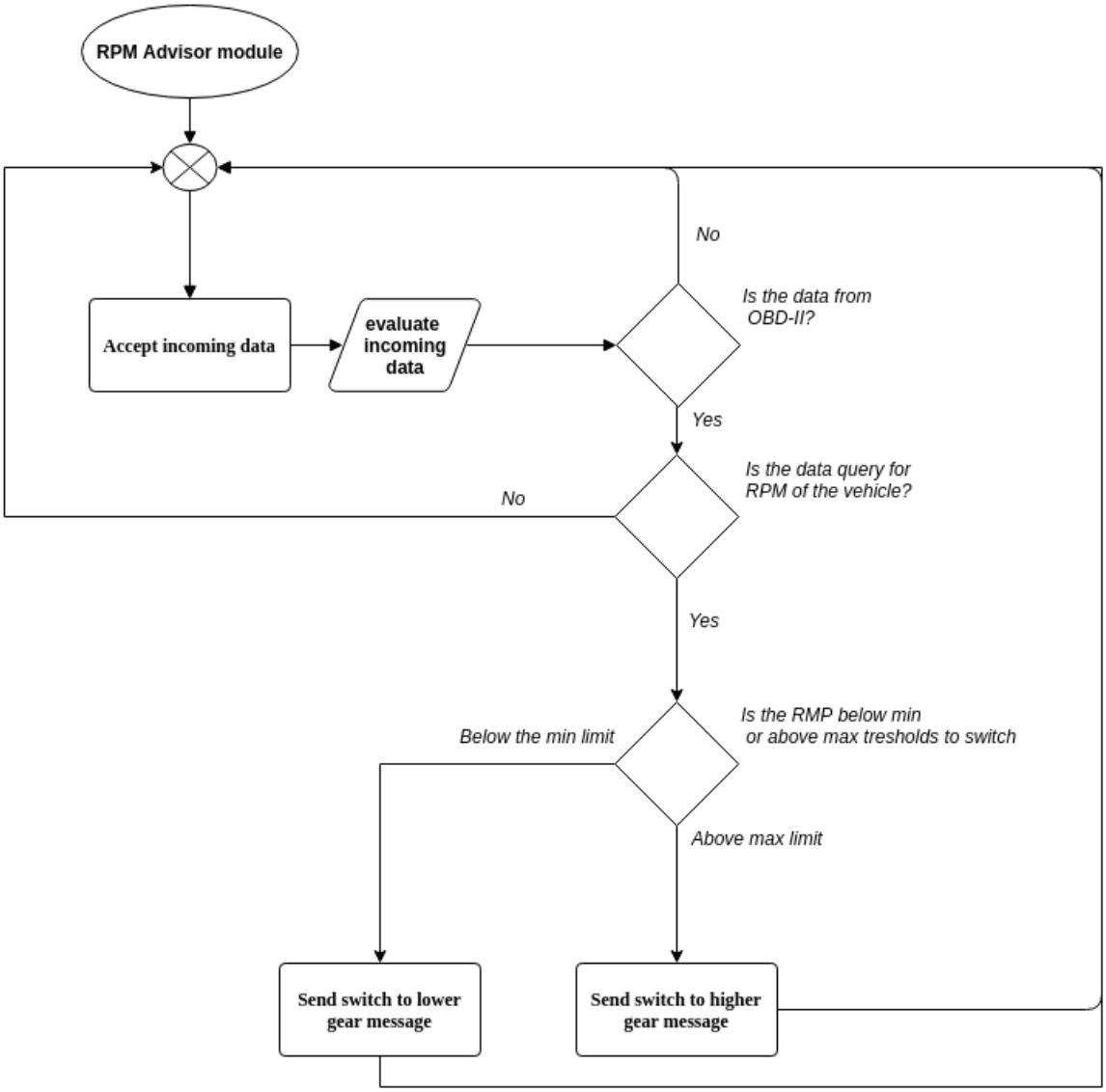



**Figure 3.5:** Activity diagram of RMP advisor module

This module have also configured IFTTT event. This event triggers a notification on mobile device which has IFTTT application installed. In the screenshots below shown as Figure 3.6 we can see what it looks like.

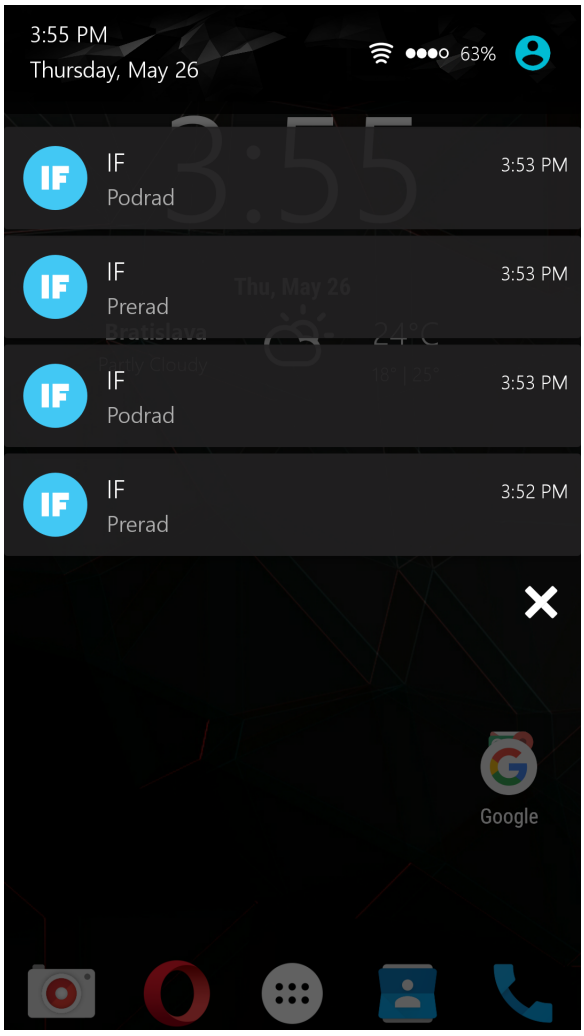



**Figure 3.6:** Screenshot of RPM advisor notifications

#### 3.4.4.2 Bulk module

Bulk module is module primarily used for grouping outputs from multiple modules into one big message. Reason for grouping messages is sending data over Internet rather in burst mode than constant stream. In bulk module settings user can choose time interval and size threshold for sending data. Whichever happens first triggers sending the message.

#### 3.4.4.3 Accident module

This module will be both accepting input data and creating new information for other modules. Electronic crash sensors found in new cars are accelerometers. They are used to determine exactly when the airbag should be deployed to prevent injury to a driver or passenger. When an impact occurs, it results in vehicle deceleration, which is sensed by the accelerometer. This change is monitored by electronic control unit, which sends a signal to trigger the airbag. Not to rely on car accelerometer sensors the application will be polling LSM303 accelerometer sensor attached to the device. Problem with car accelerometer sensors is missing common interface to communicate with airbag/ECU system. If it is even

possible then every car manufacturer has proprietary system in place, which would result in buying different specifications for every make, maybe even model. As we are not physicists we could not successfully determine at which value from accelerometer is right to register accident. But we created this module that in settings can be found threshold value which says that if accelerometer module registers acceleration over threshold, accident module will be triggered and will compose a SOS message to be sent to emergency dispatch center as shown in Figure 3.7 This message will be composed of latest GPS position, maximal g-forces registered and medical information about driver. Medical information about driver, or in other words custom message is also configurable through settings.

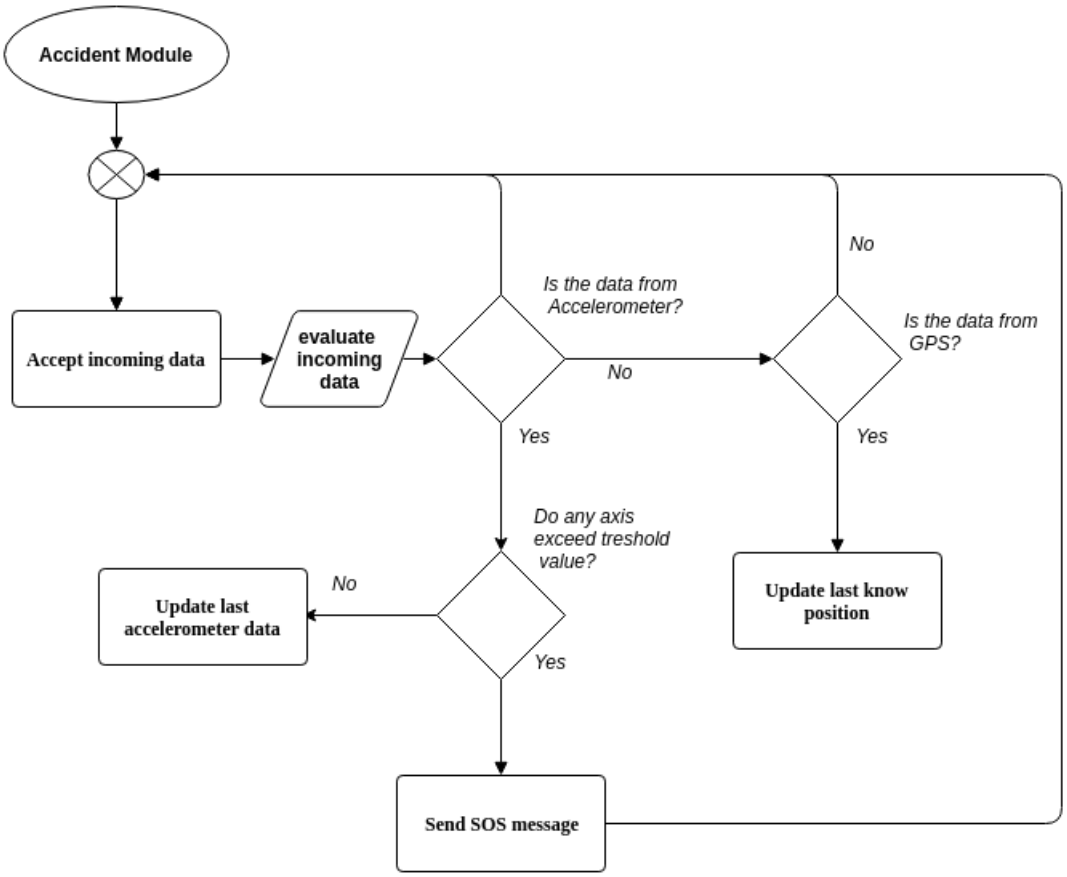



**Figure 3.7:** Activity diagram of Accident module

### 3.4.5 Data-Logger routes

Data-Logger routes are designed for interconnecting data streams from modules. These routes are designed so they would fulfill given assignment. Exact definition how routes are set can be found in a configuration file. Main benefit of having these routes in one place and so easy configurable is that you can change complete outcome of the application by changing routes and redefining them to suit your needs. All defined routes for our Data-Logger solution are provided in Table 3.3

| Source Module ID or type | Destination module IDs or types |
|---|---|
| obd | rpm, bulk, redisModule |
| rpm | ifttt |
| acc | bulk,accident, redisModule |
| gps | bulk, accident, redisModule |
| accident | sms |
| bulk | rabbitmqModule |

**Table 3.3:** Routes defined for Data-Logger of this thesis

### 3.4.6 Main application

Main application is the main part of code which is directly run by Node.js. Architecture of the main application can be described as loosely coupled system, where main application has and makes use of little to no knowledge of the Data-Logger framework. This allows for extensibility in design, a new module for framework implementing blank module interface can be written to replace a current modules without requiring a change of the main application. The new and old modules can be interchanged freely. Whole main application can be divided by features. In this section of this work every feature is described separately to provide more detail.

#### 3.4.6.1 Web Server

To provide web application for the user to be able to configure Data-Logger framework, server had to be created. This server works at port 8000 at localhost as to not confuse the server address with regular website. To connect to server, you have to be connected to the raspberry, either via LAN cable or to the wireless AP created automatically by the raspberry device. Password for this access point can be found in configuration file of hostapd in resources under name `hostapd.conf`. Server supports web-sockets to communicate real-time with Web Application, also for configuring module settings.

#### 3.4.6.2 Web Socket

For web application to display real time data produced by Data-Logger modules web-sockets are used. For every module which produces data, main application will emit the information over web-socket to every connected client. After connection client automatically starts listening for emitted info and display it accordingly. Through web-sockets it is possible to send new configuration to the server. Validating new configuration is burden for the web application as web-sockets assumes that configuration is written correctly.

#### 3.4.6.3 Web Application

Once you are connected to the raspberry, open your favorite web browser and enter default IP address of raspberry device which, if you did not change original configuration is `http://10.1.1.100:8000/`. After opening web application you will be presented with simple layout. At the top of website, navigation bar is displayed for switching between configurations of each module, configuration of routes and main page. On main page you will be provided with real time data generated by the modules. In module pages current configuration is displayed which is queried by get request to the server. Server then reads current configuration from file, parsing it and sending it back to web application. Module settings can be changed by clicking configure button. A form will be presented which will allow to add, delete and configure settings. After finishing there is the possibility to either confirm or cancel new settings. When confirmed, web application will check for syntax correctness. Web-socket will be used for sending new configuration to the main application. After getting data to the server it's passed to the Data-Logger framework to update changes to the running Data-Logger and write them to the configuration file for changes to be persistent.

# 4. Implementation

In this chapter of bachelor thesis technical implementation details regarding our solution are provided.

## 4.1 Overview

To fulfill assignment we have chosen to implement it mainly in JavaScript programming language, which provides various benefits.

- Application is operating system independent
- For back-end solution we used Node.js, which is also implemented in JavaScript. So there is no need to know another programming language by the users
- The npm package registry holds over 250,000 reusable packages of reusable code.
- Accessing Web application is possible from every web-friendly device.

As you can see there is a lot of support surrounding Node.js environment which provides a lot of reusable code. To provide language overview of our application it has over 70% of the code written in JavaScript, rest is HTML and CSS.

## 4.2 Data-Logger

Data-Logger is a standalone module. When Data-Logger is required whole module is instantiated. For correct configuration it is needed to call configure function with JavaScript Object as a argument which is passed by reading JSON configuration file from main application.

### 4.2.1 Modules

To process the file, modules have to be created by instantiating them with module options, which are then stored in internal key value object where keys are unique ids supplied by the

configuration file. As this framework is primarily intended for fellow computer scientists and power users, Data-Logger does not implement any failsafe for not overwriting modules with same ID. This happens to be a mistake to forgo this failsafe, and will be reported as one of many things to be improved in continued work after this thesis.

### 4.2.2 Routes

After modules are instantiated it is time for creating data flows defined by routes. To standardize data exchange format every message must be JavaScript Object. Every message traveling between modules must be composed of these 2 key mapped values, firstly its header where the mandatory key is id, which value is referring to the module in which message originated. Other properties of message header are optional and can be configured with main config file. Second mandatory key mapped value is body, in which is stored actual data payload created by original module. Overview of message format is provide in Table 4.1.

| Data-Logger Message | | |
|---|---|---|
| Header | | Body |
| Module ID | [Optional values] | Data payload |

**Table 4.1:** Format of Data-Logger message

To better understand what interfaces must be implemented by the modules so they could create data or accept data we provide them in specification of interfaces below.

### 4.2.3 Interface of data creating module

For data created by the module to be properly handled by Data-Logger module object must be a instance of the EventEmmiter class which exposes an eventEmitter.on() functions that allows for registering event listeners to that particular object. When object emits a specifically named event every function attached by on() function are called synchronously. So for data to be picked up the Data-Logger module must emit ‘data’ event which argument is created data. Nothing more is mandatory to be implemented.

### 4.2.4 Interface of data accepting module

Module which will be accepting data must be an instance of Stream Writable abstract class designed to be extended with an underlying implementation of the ._write method. This

method must be implemented to accept data passing to the underlying resource in case of this thesis – module. Method _write has 3 arguments, (chunk, encoding, callback). Data intended for the module is in chunk variable.

### 4.2.5 Lower-level modules

In this section we describe specific implementations of lower-level modules. Every of these modules which needs unknown amount of time to make connection to outside service implements simple queue to buffer incoming data till it is not ready to send data to the service. After successful connection is made, firstly data from queue are emptied and then normal messages are followed.

#### 4.2.5.1 Blank module

Blank module is designed to be used as default module for developing new modules. It combines data accepting and sending interface firstly by assigning this value from EventEmitter class which allows us to define and listen for events. Data accepting part is implemented by prototypal inheritance of Writable stream class, which will enable to address whole module as a stream object in which can be written to.

#### 4.2.5.2 Accelerometer module

Accelerometer is module developed for communication with Adafruit 10DOF Breakout board over $I^2C$ board. It is dependent on stand alone module application which was implemented for communicating with LSM303 accelerometer chips. This application provides read function which will query the device for current state of accelerometer readings and then report it back in a JavaScript object where values are bind to the axis from which they were read e.g.

```
{"x": x_axis_value, "y": y_axis_value, "z":z_axis_value}
```

Module provides a way how to set up accelerometer differently than default values(readings of 3 axis, 400Hz update rate), but to successfully configure these options, documentation to LSM303 should be read and our code studied, it is not a trivial task.

#### 4.2.5.3 GPS module

GPS module implements serial port access to the nmea port of Adafruit FONA 3G. For parsing read NMEA sentences module from NPM.js library is used called GPS.js. This library parses NMEA sentences into one JavaScript object which then can be queried about current position, fix state, how many satellites is connected, etc.

#### 4.2.5.4 OBD-II module

OBD module implements serial port access for ELM-USB device to connect and query OBD-II port. As every query for parameters must be executed and awaited reply, module implements queue for commands which are yet to be send to the ELM-USB. After command is pushed into the queue, event `sendCmd` is dispatched, which if cmdInProgress is false writes next query to the ELM-USB and awaits reply. When reply arrives it is parsed and all possible outcomes are processed. One specific outcome is worth to mention is when `UNABLE TO CONNECT` it signifies that ELM-USB was not able to connect to the OBD-II port for unknown reason(older car, wrong state of OBD-II port, etc.) After this outcome is detected, module will attempt repeatedly to send last command until it gets other answer than `UNALBE TO CONNECT`. In testing this module, we observed mainly on older cars sometime OBD-II port was not responsive for first time to query the data but after repeated tries, it began to answer normally with data. Thus we developed repeating of last command until we get answer. Time which should be waited before next try is configurable by 'failedDelay' setting. After initializing module a sequence of commands are send to ELM-USB to select automatic protocol selection, set ELM-USB to correct formating of responses to enable faster querying and last command to be send is to query device which OBD-II PIDs commands are supported. This is achieved by first sending `0100` command which gets bit encoded response of supported first 32 PID commands in `01` (real-time) mode. If last of this bit is set to 1, it means that support for next set of 32 commands can be queried. By this method whole bitmap of every supported command in real-time mode is queried and interval loop is established to query every supported command. By searching the Internet we were able to find how to parse some of this queries outputs and are provided in `obd.pids.js` file with short description.

#### 4.2.5.5 Redis module

Redis is module which serves as one of many possible endpoints of dataflow. Redis is module which handles database connection to Redis. For every accepted data, based on message header counter is created in the database, to enable us save each new data to be saved by unique identification key. To query last data from concrete module, it is needed to know module ID. By querying module ID from redis we will get last number used to create unique key, if we than concatenate module ID with this last number we get unique key for last data saved by redis module. By correctly configuring snapshotting intervals, we are able to continue in generating unique counters in case of power-loss or reboot of the device, because last snapshot is automatically loaded by redis when database is started on boot up.

#### 4.2.5.6 RabbitMQ module

RabbitMQ is module which handles AMQP protocol connection to the RabbitMQ service hosted on virtual private server on the Internet. Upon successful connection to this service, for every message with unique id queue is created. This provide separated queues to process on server side of the RabbitMQ service which enable server independent processing of data. Connection to the RabbitMQ service is password and user protected and can be used over SSL/TLS connection providing needed security of transferring data, however this feature is not yet implemented due lack of time and bigger scope of this thesis.

#### 4.2.5.7 SMS module

SMS module is developed to enable our application to send distress SMS messages to emergency dispatch center. This module is not developed to be used as data sending to SMS messages universal module, it is rather tailored directly for reporting accidents from data generated by accident module. SOS reporting is divided in three messages, first describes medical condition which can be set by configuration file, then last known gps location and g-force which was recorded during threshold overstep. Every message is sent trough AT command in serial port, after which special character ‘
x1A’ is used to indicate end of message. Module uses slightly different queue design as OBD-II module.

#### 4.2.5.8 IFTTT module

This module is design to trigger Maker dependent recipes at web service IFTTT. Events are triggered by making POST request to special URI address with token which is obtained from IFTTT Maker channel. In this POST message body consists of JSON object which can have 3 elements, keys are `value1,value2,value3`. This naming is not optional and is enforced by IFTTT implementation, however one can put any data into values mapped to these keys. Event name which to trigger is incorporated together with token into URI to which POST request is made. Module obtains event name by 2 methods, either in message header iftttEvent value is declared or module identification name is used.

#### 4.2.5.9 Console module

This module is debug module which displays every message routed to this module on standard console output of Node.js application. Only thing that can be configured is “messageHeader” which is automatically prefixed to console output message together with module identification obtained from message header.

#### 4.2.5.10 Time module

Time module is simple module which emits Date object through interval. Time is configurable by setting sampleRate in configuration file.

### 4.2.6 Higher-level modules

In this section we provide implementation details for higher-level modules of this application. This modules implements some degree of data analysis logic.

#### 4.2.6.1 Accident module

Accident module gathers data from two low-level modules, accelerometer module and gps module. These modules provides information necessary for accident reporting. Module establishes last known address from gps data, and last known accelerometer data. Before overwriting last accelerometer data, threshold check is ran to identify if accident happened. Threshold is configurable by configuration file. After accident is detected message is composed by providing information about the driver, which can be also configured, last known position and maximal g-forces which were registered till the accident detection.

#### 4.2.6.2 RPM advisor module

RPM advisor module is module to provide proof of data filtering from one low-level module. OBD-II module is reporting every supported parameter in connected vehicle which is not needed for RPM advisor. Module implements data filtering based on PID code which is attached to the message payload from OBD-II module. After RPM(Revolutions per minute) are obtained thresholds are checked for minimal and maximal RPMs. If threshold is below minimum or above then maximum message is emitted with advice to switch gears correctly. Module has defined ifttt Event name in message header which allows IFTTT module to correctly trigger event.

#### 4.2.6.3 Bulk module

Bulk module is developed for burst mode of sending information over the Internet to the Servers. Bulk module accumulates data sent to the module and when one of the triggers happens it send whole data as JSON array. Triggers can be twofold. First, amount of time during which data is collected and second size of collected data, these triggers are checked upon every arrival of new data. To properly calculate size of collected data additional library `sizeof` is used.

### 4.2.7 Data-Logger routes

For every data producing module, which is found on key side of key-value pairs in routes is encapsulated in InputStream class. This class provides interface listener for event ‘data’ which signals InputStream that data is ready to be sent. After event is processed, information is pushed into readable stream. Creating this readable stream allows Data-Logger to

implement piping logic of streams, where every route defined is handled by piping together readable stream(data source) to writable stream(data sink). After this piping is done, data can flow freely without being slowed through Data-Logger handling. This piping occurs for every source and sink defined in configuration. If routing by type occurs, Data-Logger internally knows every type of module which was loaded. All needed combinations are iterated and routing is executed by piping all sources to all sinks respectively by types.

## 4.3 Main application

Main application is where internal logic happens, HTTP server is created, configuration file is loaded through asynchronous file reading into JSON string and then parsed into JavaScript object. Data-Logger is instantiated with configuration object as argument. In this section we provide information about implementation details of main application and subcategories.

### 4.3.1 Web server

Application creates HTTP server which listens on port `8000` and instantiates web-socket server using Socket.io library. Application extends AJAX calls trough `/api` extension on server address. This enables web application to load configuration file asynchronously. For web styling SASS extension is used together with Bootstrap front-end framework. HTML Template engine is used for easier HTML page creation, it is called Swig and it is very similar to Django HTML template engine. Server supports angular framework for enhancing DOM capabilities and to build single-page web application. Main application is implemented to listen for terminal commands send to Node.js, after registering command to quit Node.js Data-Logger is properly shutdown, by calling internal function, so every module would have a chance to finish writing and sending messages.

### 4.3.2 Web Socket

Sockets listen for `configureModule` event which awaits JavaScript configuration object which is then passed to the Data-Logger framework. Same is happening with `configureRoute`, but rather module configuration, whole routes configuration is passed. For every module which has implemented data sending interface, in another words which has EventEmmiter registering function on(), real-time data listener is hooked up. Upon firing data event, data is not only processed by Data-Logger but is also send to Web application

front-end.

### 4.3.3 Web Application

Web application is implemented as Single Page Application(SPA). It is built with Angular JavaScript framework and Angular routing. Once a web request for web application is handled by the server, client load all resources for web application to work, so no other requests are needed. Web application implements routing for every module by adding `/module/:ID` to the link of the web application. By accessing this page we are able to configure every module separately. Configuring is handled by changing displayed configuration to form, in which single text-area has old configuration. After editing this configuration and confirming to apply changes, new configuration is evaluated for syntax mistakes as configuration needs to be in valid JSON format. If syntax error is found then configuration is not changed. If there are no syntax errors, configuration changes and through web sockets new configuration is send into Data-Logger. Overview page is created to show real-time data flow of every module. If module's table is red, it means no data was displayed yes, if it is green, message was registered from that particular module.

# 5. Conclusion

In this chapter we summarize what was achieved in our solution and compare it to the requirements and then suggests what can be done in future work with this bachelor thesis.

## 5.1 Result

In this thesis our goal was to provide embedded device which integrates multiple data sources into system which can distribute collected information to various services. All peripheral devices were successfully connected to the device and are operating. Accelerometer data are recored through $I^2C$ protocol, cellular modem is capable of connecting to the Internet and sending SMS messages, Wi-Fi extension dongle is automatically used to create Software Access Point to which anybody with password can connect. OBD-II port is interconnected with device through ELM-USB cable and is properly configured through our application to connect to the car.

An application was developed which allows collecting of data from multiple sources. Application is configurable, modular and modules are interchangeable. Internal communication between modules is designed so modules can send and accept data. In routes modules are selectable by types, which allows to define large dataflows with simple routing. Modules are provided for gathering sensoric data from accelerometer, OBD-II port of vehicle, NMEA data from GPS sensor and time from OS. Modules for sending data are: sending SMS messages, sending messages over AMQP protocol triggering IFTTT events, storing data into Redis database and displaying data to the standard output of Node.js application. By combing these low-level modules and adding logic high-level modules are built. Modules which we developed are for reporting accidents with SMS messages, advising drivers when to shift gears based on Revolutions per Minute(RPM) of the car, sending data in bulk mode over cellular network instead of constant stream. Routes for correct functionality of the application are also provided.

Web application is accessible through wireless access point and is used for configuring modules and displaying real-time data from modules. Application upon correct installation

on embedded device works and is capable of monitoring various inputs. This information is collected, distributed and made available for different services.

## 5.2 Future work

Main goal of future work is to make available our Data-Logger framework for everyone, i.e. create an installation module in the npm.js module directory. After Data-Logger is available, start producing application modules for various sensors, devices and services based on popular demand. Create a guide or wiki pages for better understanding of how Data-Logger works and to attract additional developers to extend it. Provide fail-safe mechanism for declaring multiple modules with same identification name to handle this error.

In modules there are even more possibilities what could be done so we name a few. RabbitMQ module should be implemented with SSL/TLS encryption to ensure data privacy when sending messages to server on the Internet. Implement additional modules to gather data from all sensors on Adafruit 10DOF breakout board as this thesis implements module for accelerometer sensor. Design more high-level modules for business logic. Main application could be password protected as are consumer-grade routers, API provided by server should have authentication method, so no sensitive information can be obtained without authorization. Create an infrastructure for analyzing data gathered by developed device, with aspect of Big Data in mind. Infrastructure should be scalable, secure and provide direct means for analyzing contents of gathered information. And lastly as infrastructure would be meaningless if there would not be enough connected cars, research how to make embedded device smaller and cheaper which would allow manufacturing on larger scale.

# Glossary

**CAN** Controller Area Network. 24, 25, 27

**ECU** Electronic Control Unit. 23, 24

**EPA** Environmental Protection Agency. 25

**GPIO** General-purpose input/output. 33

**$I^2C$** Inter-Integrated Circuit. 16, 17, 30, 33, 37, 38, 54, 60

**IoT** Internet of Things. 14, 15

**LAN** Local Area Network. 24

**OBD-II** On-Board Diagnostics port. 13, 26, 27, 30, 31, 38, 55, 56, 60

**OEM** Original equipment manufacturer. 26

**PIDs** Parameter IDs. 27, 55

**SAE** Society of Automotive Engineers. 25

**SBC** Single-board computer. 17, 19–21, 33

**V8** V8 JavaScript Engine. 28

# References


[1] ABIDI, Hatem. . *Cours Systèmes Embarqués: BUS I2C*. [online]. : . [visited on 20.4.2016]. URL: http://www.technologuepro.com/cours-systemes-embarques/cours-systemes-embarques-Bus-I2C.htm.

[2] ADA, Lady. . *Adafruit FONA 3G Cellular Breakout - European version*. [online]. : . [visited on 20.4.2016]. URL: https://learn.adafruit.com/adafruit-fona-3g-cellular-gps-breakout/.

[3] —, . *Bluetooth / WiFi Combination USB Dongle*. [online]. : . [visited on 20.4.2016]. URL: https://www.adafruit.com/products/2649.

[4] —, . *Intel® Edison w/ Mini Breakout Board*. [online]. : . [visited on 20.4.2016]. URL: https://www.adafruit.com/products/2111.

[5] ALEE, N. - RAHMAN, M. - AHMAD, R. B. "Performance comparison of single board computer: a case study of kernel on arm architecture", in *Computer Science Education (ICCSE), 2011 6th International Conference on* : Aug. 2011. S. 521–524. DOI: 10.1109/ICCSE.2011.6028693.

[6] ALMEIDA, Tatiana De. . *About Automatic*. [online]. : https://www.automatic.com. [visited on 20.4.2016]. URL: https://www.automatic.com/press/.

[7] AUDISIO, G. "The birth of the cyber tyre", *IEEE Solid-State Circuits Magazine*, vol. 2, no. 4, s. 16–21, Fall 2010, ISSN: 1943-0582. DOI: 10.1109/MSSC.2010.938647.

[8] AUTOMATIC. . *How Automatic works*. [online]. : https://www.automatic.com. [visited on 20.4.2016]. URL: https://www.automatic.com/how-automatic-works/.

[9] BEAGLEBOARD.ORG. . *BeagleBone Black*. [online]. : . [visited on 20.4.2016]. URL: http://beagleboard.org/static/ti/product_detail_black_lg.jpg.

[10] BEAUTIFUL, Information. . *Codebases, Milions of lines of code*. [online]. : . [visited on 20.4.2016]. URL: http://www.informationisbeautiful.net/visualizations/million-lines-of-code/ (visited on 04/20/2016).

[11] CHEN, H. - TIAN, J. "Research on the controller area network", in *Networking and Digital Society, 2009. ICNDS '09. International Conference on*, vol. 2 : May 2009. S. 251–254. DOI: 10.1109/ICNDS.2009.142.

[12] CORPORATION, Intel. . *Edison Overview*. [online]. : . [visited on 20.4.2016]. URL: http://download.intel.com/support/edison/sb/edison101presentationfromidfforthecommunity.pdf.

[13] DR. OVIDIU VERMESAN, Dr. Peter Friess 2013. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems* : River Publishers, 2013. . ISBN: 9788792982964.

[14] ERNIOTTI. . *CAN-Bus levels*. [online]. : . [visited on 20.4.2016]. URL: https://commons.wikimedia.org/wiki/File:CAN-Bus-frame_in_base_format_without_stuffbits.png.

[15] FARSI, M. - RATCLIFF, K. - BARBOSA, M. "An overview of controller area network", *Computing Control Engineering Journal*, vol. 10, no. 3, s. 113–120, Jun. 1999, ISSN: 0956-3385. DOI: 10.1049/cce:19990304.

[16] FOUNDATION, Raspberry Pi. . *What is a Raspberry Pi*. [online]. : . [visited on 20.4.2016]. URL: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/.

[17] GOOGLE. . *Chrome V8 Introduction*. [online]. : . [visited on 20.4.2016]. URL: https://developers.google.com/v8/intro.

[18] GU, X. F. - YANG, L. - WU, S. "A real-time stream system based on node.js", in *Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2014 11th International Computer Conference on* : Dec. 2014. S. 479–482. DOI: 10.1109/ICCWAMTIP.2014.7073454.

[19] HTTP://I2C.INFO/. . *I2C Bus Specification*. [online]. : http://i2c.info/. [visited on 20.4.2016]. URL: http://i2c.info/i2c-bus-specification.

[20] *I2c-bus specification and user manual*, Rev. 6, NXP Semiconductors : Apr. 2014.

[21] JAMES, Mike. . *What Is Asynchronous Programming?*. [online]. : . [visited on 20.4.2016]. URL: http://www.i-programmer.info/programming/theory/6040-what-is-asynchronous-programming.html.

[22] MI-JINKIM, Yun-Sik Yu, "A study on in-vehicle diagnosis system using obd-ii with navigation", *International Journal of Computer Science and Network Security*, vol. 10, no. 9, s. 136–140, Sep. 2010, ISSN: 1738-7906.

[23] KRIDNER, Jason. . *What is Beagle?*. [online]. : . [visited on 20.4.2016]. URL: http://beagleboard.org/brief.

[24] KUGELSTADT, Thomas. . *SIGNAL CHAIN BASICS (Part 32): Digital interfaces (con't) – The I2C Bus*. [online]. : . [visited on 20.4.2016]. URL: http://www.planetanalog.com/document.asp?doc_id=527900&site=planetanalog&.

[25] LEEN, G. - HEFFERNAN, D. "Expanding automotive electronic systems", *Computer*, vol. 35, no. 1, s. 88–93, Jan. 2002, ISSN: 0018-9162. DOI: 10.1109/2.976923.

[26] LEEN, G. - HEFFERNAN, D. - DUNNE, A. "Digital networks in the automotive vehicle", *Computing Control Engineering Journal*, vol. 10, no. 6, s. 257–266, Dec. 1999, ISSN: 0956-3385. DOI: 10.1049/cce:19990604.

[27] LUCASBOSCH. . *Raspberry Pi B+ top.jpg*. [online]. : . [visited on 20.4.2016]. URL: https://upload.wikimedia.org/wikipedia/commons/6/6f/Raspberry_Pi_B+_top.jpg.

[28] "News briefs", *Computer*, vol. 47, no. 7, s. 16–21, Jul. 2014, ISSN: 0018-9162. DOI: 10.1109/MC.2014.189.

[29] OKMIJ. . *Monadic i/o and UNIX shell programming*. [online]. : . [visited on 20.4.2016]. URL: http://okmij.org/ftp/Computation/monadic-shell.html.

[30] PTC. 2015. *Reducing Mean Time to Repair by 50 Percent with SmartConnect™*. [online]. : http://www.ptc.com/, 2015. [visited on 20.4.2016]. URL: http://www.ptc.com/internet-of-things/customer-success/varian-medical-systems/thank-you.

[31] QUINLAN, Daniel. . *Filesystem Hierarchy Standard*. [online]. : . [visited on 20.4.2016]. URL: http://www.pathname.com/fhs/2.2/fhs-3.15.html.

[32] . . *Raspberry Pi*. [online]. : . [visited on 20.4.2016]. URL: https://wiki.archlinux.org/index.php/Raspberry_Pi.

[33] REDIS. . *Documentation*. [online]. : . [visited on 20.4.2016]. URL: http://redis.io/documentation.

[34] . . *RPi SD cards*. [online]. : . [visited on 20.4.2016]. URL: http://elinux.org/RPi_SD_cards.

[35] RUI, J. - DANPENG, S. "Architecture design of the internet of things based on cloud computing", in *2015 Seventh International Conference on Measuring Technology and Mechatronics Automation* : Jun. 2015. S. 206–209. DOI: 10.1109/ICMTMA.2015.57.

[36] SECONS. . *Elm-usb*. [online]. : . [visited on 20.4.2016]. URL: http://www.obdtester.com/elm-usb.

[37] —, . *Identification commands*. [online]. : . [visited on 20.4.2016]. URL: http://www.obdtester.com/elm-usb-commands.

[38] SIMCOM. . *AT Command Set*. [online]. : . [visited on 20.4.2016]. URL: https://cdn-shop.adafruit.com/datasheets/SIMCOM_SIM5320_ATC_EN_V2.02.pdf.

[39] . . *Software access point*. [online]. : . [visited on 20.4.2016]. URL: https://wiki.archlinux.org/index.php/Software_access_point (visited on 04/20/2016).

[40] STANDARDIZATION, International Organization. 2014. *Road vehicles – Diagnostic communication over Controller Area Network*. [online]. : http://www.iso.org/, 2014. [visited on 20.4.2016]. URL: http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=46045.

[41] SZALAY, Z. - KÁNYA, Z. - LENGYEL, L. - EKLER, P. - UJJ, T. - BALOGH, T. - CHARAF, H. "Ict in road vehicles #2014; reliable vehicle sensor information from obd versus can", in *Models and Technologies for Intelligent Transportation Systems (MT-ITS), 2015 International Conference on* : Jun. 2015. S. 469–476. DOI: 10.1109/MTITS.2015.7223296.

[42] TILKOV, S. - VINOSKI, S. "Node.js: using javascript to build high-performance network programs", *IEEE Internet Computing*, vol. 14, no. 6, s. 80–83, Nov. 2010, ISSN: 1089-7801. DOI: 10.1109/MIC.2010.145.

[43] . . *tmpfs*. [online]. : . [visited on 20.4.2016]. URL: https://wiki.archlinux.org/index.php/Tmpfs.

[44] TOWNSEND, Kevin. . *Adafruit 10-DOF - L3GD20H + LSM303 + BMP180*. [online]. : . [visited on 20.4.2016]. URL: https://learn.adafruit.com/adafruit-10-dof-imu-breakout-lsm303-l3gd20-bmp180/.

[45] TRUE, Agamand The. . *Understanding Node.js*. [online]. : . [visited on 20.4.2016]. URL: https://dotnetadil.wordpress.com/2012/01/31/understanding-node-js/.

[46] . . *VI Monitor Features*. [online]. : . [visited on 20.4.2016]. URL: http://www.vi-performance.com/?q=node/2.

# A. Automatic Hardware Specification

- OBD-II (conforms to J1962 standard), works with most cars since ’96
- Weight and Dimensions: 1.65 x 1.96 x 0.78in (42 x 50 x 20mm) 0.84oz (23.75g)
- Temperature Tolerance: Operating: -40 °F (-40 °C) to 158 °F (70 °C), Resting: -40 °F (-40 °C) to 185 °F (85 °C)
- Wireless: Bluetooth 4.0 Dual Mode (EDR and BLE), Made for iPhone (MFi) Certified
- Accelerometer: Frequency: 100Hz, Precision: 0.012G
- Security: Wireless encryption: 128-bit AES, Signed binary enforcement: 1024-bit RSA
- Built-in GPS: GNSS engine for GPS/QZSS, Logs trip routes with no phone present
- Audio Capabilities: Volume: 80db at 10cm at 2.5kHz
- Firmware: Over-the-air updates via smartphone

# B. VI Monitor abilities

- Monitor parameters such as RPM, Speed, Throttle Position, Intake Manifold Pressure, Water Temperature, Air Fuel Ratios (lambda), Air Flow Rate, Ignition Advance Fuel Pressure, and many more.
- Perform braking and acceleration tests such as 0-60, 1/4 mile and 0-60-0 to measure your car's true performance. Most tests can be G-triggered for unparalleled accuracy. Each test is recorded for future comparison.
- Avoid putting points on your license with adjustable speed warnings. Use the RPM warnings in conjunction with the adjustable Shift Light feature to get the most from your engines performance.
- Highly accurate G-Sensor with built-in damping monitors acceleration, braking and cornering G-Forces. Also records maximum G-readings.
- Record over 500 hours of engine and performance data on any parameters for review. Then upload the data to your computer for comparison. Ideal for measuring the effectiveness of modifications and recording drivers performance.
- Got a Engine Warning Light, but dont know why? - VI gives you the fault code number and a description of the problem, giving you more information to take to your garage or tuner.
- Simple stop/start timing feature allows you to record your times for later comparison on a computer.
- VI allows you to reset fault codes yourself, It also maintains a complete MIL stats history, including time and distance since the engine warning light was activated or rest.
- Using a Built-in Virtual Dynamomter, VI can test your vehicles true net horsepower in real world conditions.

# Bluetooth / WiFi Combination USB Technical Details

- Ralink RT5370L WiFi adapter + BCM2046B1 Bluetooth 3.0 adapter
- Full-speed Bluetooth operation with Picone and Scatternet support
- Compliant with Bluetooth 2.1 + EDR and Bluetooth V3.0 + HS
- Support for WLAN + Bluetooth coexistence for optimized performance
- IEEE 802.11b/g/n compliance
- 150 Mbps Tx (transmit) PHY data rate
- 150 Mbps Rx (transmit) PHY data rate
- USB 1.1/2.0 compliant

# C. npm packages

For clarification which packages I used during implementation, with short reasoning why I used it.

- **amqplib** – library used for making amqp 0.9.1 clients for Node.js, it has been used to create client for RabbitMQ module[4.2.5.6]
- **body-parser** – body parsing middleware, once internal part of express, now must be installed separately, middleware is used to handle POST requests
- **duplexer2** – library for creating writable and readable streams as one, used in internal structures of Data Logger.
- **express** – web application framework that provides a robust set of features for web and mobile applications.
- **express-session** – simple session middleware for Express, used in web-server middleware.
- **gps** – GPS.js is an extensible parser for NMEA sentences, used by GPS module to parse incoming data.[4.2.5.3]
- **i2c-bus** – I2C serial bus access used in accelerometer module[4.2.5.2]
- **lodash** – A modern JavaScript functional utility library delivering modularity, performance and extras. Library is used throughout whole application for correct handling with objects and arrays.
- **morgan** – HTTP request logger middleware for node.js, used to log HTTP requests on the express server.
- **node-sass-middleware** – Connect middleware for node-sass, recompile .scss or .sass files automatically for connect and express based HTTP servers.[citation needed]
- **object-sizeof** – library for computing size of a JavaScript object in bytes, used in bulk module to evaluate message size[4.2.6.3]
- **path** – this module contains utilities for handling and transforming file paths.[https://nodejs.org/docs/latest/api/path.html]

- **redis** – this is a complete and feature rich Redis client for node.js.[https://www.npmjs.com/package/redis] It is used in redis module[4.2.5.5]
- **request** – Request is designed to be the simplest way possible to make HTTP calls. Used in IFTTT module to generate POST requests to the Maker channel[4.2.5.8]
- **serialport** – Node.js library to access serial ports, used in every module which required access to serial port.
- **socket.io** – node.js real-time framework server, used to communicate through web-sockets or JSON long polling
- **swig**
  – A simple, powerful, and extendable JavaScript Template Engine. Used in HTML templates for web application.[3.4.6.3][http://paularmstrong.github.io/swig/docs/]

# D. Resources

CD which contains:

- Digital copy of this bachelor thesis
- All sources files
- All configuration files which are used in this bachelor thesis

Resources which are on CD are also available on GitHub repository `https://github.com/dejwoo/bachelor_thesis`.