

**COMENIUS UNIVERSITY IN BRATISLAVA  
FACULTY OF MATHEMATICS, PHYSICS AND  
INFORMATICS**

**INTERNET OF THINGS IN  
AUTOMOTIVE INDUSTRY**

**Bachelor's thesis**

**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: 9.2.9. Applied Informatics  
Department: Department of Applied Informatics  
Supervisor: RNDr. Jozef Šiška, mgr?  
Advisor: Mgr. Anton Pytel



Univerzita Komenského v Bratislave  
Fakulta matematiky, fyziky a informatiky

## ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:**

**Študijný program:**

aplikovaná informatika (Jednoodborové štúdium,  
magisterský II. st., denná forma)

**Študijný odbor:**

9.2.9. aplikovaná informatika

**Typ záverečnej práce:**

diplomová

**Jazyk záverečnej práce:**

slovenský

**Názov:**

**Cieľ:**

**Anotácia:**

**Vedúci:**

**Katedra:**

**Dátum zadania:**     :

**Dátum schválenia:**     :

.....  
študent

.....  
vedúci práce

## **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, 2015

.....

## **Acknowledgements**

I would like to thank my supervisor...

## **Abstrakt**

Tu je text slovenskej verzie abstraktu

**Kľúčové slová:** *slovo1, slovo2, slovo3, slovo4*

## **Abstract**

In this bachelor thesis we will show you the possibilities of connected cars, options in what advanced services a connected vehicle can provide and how you can do it in the first place. After a thoughtful study of our work anybody with computer science background will be able to build a device which will be capable of advanced services based on his/her needs.

**Keywords:** *Automotive industry, IoT, Big Data, Embedded device*

# Contents

<b>Introduction</b>	<b>11</b>
<b>1 Overview</b>	<b>12</b>
1.1 Solutions available . . . . .	12
1.1.1 Automatic . . . . .	12
1.1.2 VI Monitor . . . . .	12
1.2 Internet of Things . . . . .	13
1.2.1 Concept . . . . .	13
1.2.2 Benefits . . . . .	13
1.3 Single-board computer . . . . .	14
1.3.1 Requirements . . . . .	15
1.4 OBD-II port . . . . .	15
1.4.1 Concept . . . . .	15
1.4.2 Why is it important? . . . . .	16
1.5 Accelerometer . . . . .	16
1.5.1 General Information . . . . .	16
1.5.2 Why is it important? . . . . .	16
<b>2 Specification</b>	<b>17</b>
2.1 Requirements . . . . .	17
2.1.1 Hardware . . . . .	17
2.1.2 System . . . . .	17
2.1.3 Application . . . . .	18
2.1.4 Output . . . . .	18
<b>3 Solution</b>	<b>19</b>
3.1 Hardware . . . . .	19
3.1.1 Device . . . . .	19
3.1.2 Operating System . . . . .	20
3.2 Configuration . . . . .	21



3.2.1	Database . . . . .	21
3.2.2	Wireless Access Point . . . . .	21
3.2.3	Internet . . . . .	22
3.3	Peripherals . . . . .	23
3.3.1	Adafruit 10-DOF . . . . .	23
3.3.2	Adafruit FONA 3G Cellular + GPS . . . . .	23
3.4	Software . . . . .	24
3.4.1	Problem abstraction . . . . .	24
3.4.2	Data Logger framework . . . . .	24
3.4.2.1	Module . . . . .	24
3.4.2.2	Route . . . . .	25
3.4.3	Data Logger modules . . . . .	25
3.4.3.1	Blank module . . . . .	25
3.4.3.2	Time module . . . . .	25
3.4.3.3	Accelerometer module . . . . .	26
3.4.3.4	GPS module . . . . .	26
3.4.3.5	Accident module . . . . .	26
3.4.3.6	SMS module . . . . .	27
3.4.3.7	OBD-II module . . . . .	27
3.4.3.8	IFTTT module . . . . .	27
3.4.3.9	Redis module . . . . .	28
3.4.3.10	RPM module . . . . .	28
3.4.3.11	RabbitMQ module . . . . .	28
3.4.3.12	Console module . . . . .	28
3.4.3.13	Bulk module . . . . .	29
3.4.4	Data Logger routes . . . . .	29
3.4.5	Main application . . . . .	29
3.4.5.1	Web Server . . . . .	29
3.4.5.2	Web Socket . . . . .	29
3.4.5.3	Web Application . . . . .	30
<b>4</b>	<b>Implementation</b>	<b>31</b>
4.1	Overview . . . . .	31
4.2	Data Logger . . . . .	31
4.2.1	Modules . . . . .	31
4.2.2	Routes . . . . .	32
4.2.2.1	Interface of data creating module . . . . .	32
4.2.2.2	Interface of data accepting module . . . . .	32

<b>5 Conclusion</b>	<b>33</b>
<b>References</b>	<b>36</b>

# Introduction

Most of us who watched Knight Rider as a kid 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 we can resort to that this is only partially true, we have successful tries at completely autonomous cars(e.g Google's 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 where 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. You can not configure them to the extent as smartphones have apps. It is either you will get used to the OEM solution or go and buy whole solution from somebody else. It is like we are stuck in the 90s, where software offered to you had come on this particular hardware and you could not choose otherwise. Imagine you would buy smartphone with predefined set of applications and you could not 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

## 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.”[5]

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.[2] More detailed hardware specification can be found in the attachment chapter[5]. 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 from vehicle electronic control unit via 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.[13] For more detailed overview of capabilities of VI Monitor consult [5]

## **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 has the storage and computing power.[11] 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.[6]

### **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 enviroments and sources. Howewer, 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.[10] 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.[8] 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[4].

- 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.[8]
- US-based HydroPoint Data Systems is using the IoT in its WeatherTRAK application to enable landscape irrigation systems, sensors, and computers to communicate autonomously online to identify leaks and other potential system problems. [8]

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 won't 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 Single-board computer**

Sensor monitoring systems 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 measures are used in manual and automation control based systems[1]. The 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 and development research/systems.

### **1.3.1 Requirements**

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 connectig using built-in hardware or peripheral device. For connecting various sensors SBC must provide enough GPIO pins and support communication protocols commonly used with sensors e.g I<sup>2</sup>C or SPI

## **1.4 OBD-II port**

OBD II is an acronym for On-Board Diagnostics II, the second generation of on-board self-diagnostic equipment. On-board diagnostic capabilities are incorporated into the hardware and software of a vehicle's on-board computer to monitor virtually every component in the car. Each component is checked by a diagnostic routine to verify that it is functioning properly. If a problem or malfunction is detected, the OBD II system will alert the driver that something is wrong. The system will also store important information about any detected malfunction so that a repair technician can accurately find and fix the problem.

### **1.4.1 Concept**

The OBD-II standard specifies the type of the connector and its pinout, the messaging format and electrical signalling protocols available. OBD-II also provides a list of vehicle accessible parameters for monitoring tohether with how to encode the data for each. One of the pin in the connector provides power to the connected unit, so it could run from the vehicle battery, which simplify use of scantools, because you do not need auxiliary power. On the other hand, sometimes auxiliary power is needed, because of car malfunction which could shutdown the scan tool and lead to loss of diagnostic data. 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. OBD-II standardization was prompted by emissions requirements, and though only emission-related codes and data are required to be transmitted through it, most manufacturers have made the OBD-II Data Link Connector the only one in the vehicle through which all systems are diagnosed and programmed. OBD-II Diagnostic Trouble Codes are 4-digit, preceded by a letter: P for engine and transmission (powertrain), B for body, C for chassis, and U for network. [12]

### **1.4.2 Why is it important?**

One of the main sources of information in my thesis will be data from OBD-II diagnostic port. This data will be accessible later in the database and we can preform logical deduction on them.

## **1.5 Accelerometer**

### **1.5.1 General Information**

An accelerometer measures proper acceleration which is the acceleration it experiences relative to freefall, and is the acceleration that is felt by people and objects. Put another way, at any point in spacetime the equivalence principle guarantees the existence of a local inertial frame, and an accelerometer measures the acceleration relative to that frame.[7]As a consequence an accelerometer at rest relative to the Earth's surface will indicate approximately 1 g upwards, because any point on the earth's surface is accelerating upwards relative to a local inertial frame. To obtain the acceleration due to motion with respect to the earth, this "gravity offset" should be subtracted.

The reason for the appearance of a gravitational offset is Einstein's equivalence principle[9], which states that the effects of gravity on an object are indistinguishable from acceleration of the reference frame. When held fixed in a gravitational field by, for example, applying a ground reaction force or an equivalent upward thrust, the reference frame for an accelerometer (its own casing) accelerates upwards with respect to a free-falling reference frame. The effect of this reference frame acceleration is indistinguishable from any other acceleration experienced by the instrument. An accelerometer will read zero during free fall. This includes use in a spaceship orbiting earth, but not a (non-free) fall with air resistance where drag forces reduce the acceleration until terminal velocity is reached, at which point the device would once again indicate 1 g acceleration upwards.

### **1.5.2 Why is it important?**

For accident detection we will use accelerometer to determine if accident occurred[3]. Application will proactively monitor the occurrence of accident using accelerometer sensor. When the accelerometer sensor values exceeds the threshold value, system can act upon it.



## **2. Specification**

### **2.1 Requirements**

#### **2.1.1 Hardware**

Provide an embedded device which will be running operating system of your preference. Design how to connect all peripheral devices. Setup the device with applications to operate properly such as Internet, Wi-Fi, GPS and additional sensors. Figure out how to connect your device to the vehicle's OBD-II service port. Assemble and provide this device as a hardware part of this bachelor thesis.

#### **2.1.2 System**

Configure the system to every sensoric input so you can communicate with them. Use any protocol for communication which will suit your solution. Develop a application for collecting information from various sources, design the system to be easily configurable, without needing to change large portions of the system. System will have modular capabilities of connecting various sources of information. Develop a routing which will enable to configure which way data flows from input sources to output sinks. Design the application for modular adding of different outputs. Define a format in which are information passed from your server. Create an interface where user can configure this input functions to increase adaptivity of your system. Figure out how to send data to the server connected to the Internet. For accessing through WiFi, create RESTful web application which will user to access your web app. In web application add a configuration webpage where user can easily configure your system. Parameters of modules connected to your application will have to be configurable by the user, define them and provide a configuration method.

### **2.1.3 Application**

Provide examples which will utilize your modular application, gathering information collected from connected vehicle. The examples need to perform logic deduction from accessed data and act upon it.

### **2.1.4 Output**

Output of your bachelor thesis should be application which upon correct installation 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 of bachelor thesis I will provide a solution of my bachelor thesis problem.

### 3.1 Hardware

For the start we need device which will be capable of running operating system. There are many various platforms which provide single board computers, and after considering pros and cons I have choosen raspberry platform as most suitable because is is capable of running full linux operating system, have 4 USB slots which will be needed for peripheral devices and foremost there is a pleathora of hardware addons, sensors and boards which works with raspberry models, not to mention availablilty of information how to connect them, tutorials how to setup them. In time of writing this thesis I can say raspberry platform is the most popular platform for single board computers.

#### 3.1.1 Device

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 lacks usb ports. It has support for On-the-go USB cables but we need device which supports atleast 2 or 3 usb ports. Which we would need to solve by buying USB hubs. The best suitable device was Raspberry Pi 2, which 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 my recommendation for the reader is to go with Raspberry Pi 3 model.

### **3.1.2 Operating System**

The most suitable operating system for our needs is Linux based, because it is open-source. 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 KISS principle in mind also it is a rolling release which means it is always 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[CITATION NEEDED]. Installing and basic configuration of Arch Linux will not be covered in this thesis, as I would repeat myself. Arch Linux has one of the best documentation online, so if you 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 I will cover configuration of various applications and system properties to set up our device properly for our needs, however it will not be complete configuration, as peripherals connected to this device will also need some degree of setup and configuration I will cover that in their respective sections.

### 3.2.1 Database

For storing information on developing 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 vary 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.[Citation needed] In spite of availability of different micro SD cards, everyone has limited read/write cycles after which card become 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 configuring we could point database to create this file in a special directory. On Linux it is called /tmp[citation needed], this direcorey is allocated not on 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 votelatile[citation needed]. My solution for database is to use special database which sets the database in RAM memory but it is configurable to save snapshots(copies) to the micro SD card. Database is called Redis. Redis is an open source, in-memory data structure store, used as database, cache and message broker.[citation needed] Sample configuration on which database runs on our device will be provided with other configurations on GitHub repository under name “redis.conf” in folder “confs”.

### 3.2.2 Wireless Access Point

To automaticly create wireless access point when the device boots up we need several configurations and applications. Firstly the compatibility of device. To create software access point you will need nl80211 compatible wireless device[Citation needed], which support AP operating mode. Application which will be needed is hostapd, iw and dhcpcd. In the confs folder on github you can find configuration file for hostapd which needs to be copied into /etc/hostapd/, and a systemctl rule, which needs to be copied

into `/lib/systemd/system/`. After that reload `systemctl` daemon, 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 I am 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 again on GitHub repository in `conf/ppp` folder. Another approach is with `wvdial`, which configuration can also be found in repository. At last FONA 3G is supported by QMI modem protocol. For dialing with `qmi` one needs to install `libqmi` and `net-tools` packages. You will need to know to which `apn` you need to connect from your mobile internet service provider. After that replace `APN` in `qmi-network.conf` from `config` folder at GitHub, 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 your SIM card is PIN locked you will need to unlock it manually through `AT` command or with help from `mmcli` command.

## 3.3 Peripherals

For providing proof of concept device which is capable of gathering information from various sources I will connect to the device numerous peripherals which will generate information. In upcoming subsections I will describe them and provide configuration instructions if necessary. I believe that connecting this peripherals is such trivia 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 I suggest to find some guide on the internet for correct connections to be made.

### 3.3.1 Adafruit 10-DOF

This inertial-measurement-unit combines 3 sensors to give you 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. Most will be pretty happy with just the plain I2C interfacing, but we also break out the data ready and interrupt pins, so advanced users can interface with if they choose.[Citation needed]. As for configuration you will need packages `i2c-tools` and `lm_sensors`, then get config `config.txt` from thesis repository and copy it to `/boot/config.txt`, enable modules `i2c-dev`, `i2c-bcm2708` in `/etc/modules-load.d/raspberrypi.conf`. Thats it, now with command `i2cdetect -y 0` it should work. if not follow troubleshoot documentation on raspberry pi on archlinux wiki.

### 3.3.2 Adafruit FONA 3G Cellular + GPS

The FONA 3G has better coverage, GSM backwards-compatibility and even sports a built-in GPS module for geolocation and asset tracking. This all-in-one cellular phone module with that lets you add location-tracking, voice, text, SMS and data to your project in a single breakout.[citation needed] As for configuration 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. After connecting modem through usb to the raspberry, kernel will create 4 serial ports(debug,nmea,serial,modem). After starting gps chip with correct command `at+cgps=1` at serial port nmea will start to push gps data on nmea port at 115200 baud rate.

## **3.4 Software**

### **3.4.1 Problem abstraction**

To solve given task, lets first look at data gathering and what can be done to optimize process of collecting information from often vary 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), lets develop a framework. 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**

This framework will be a standalone module which will be incorporated in the final application. 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. Data Logger is configurable through internal function and by file which serves for storing configurations of each model and routes. This is the first version of the framework and will probably undergo some more development.

#### **3.4.2.1 Module**

Module is basic structure for every item. It is an encapsulating class for every information creating or sending application/module. By this framework is able to objectify every application be it different, and extract data from generating one or send data to accepting one. By this single 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 modules to parse/get data from the source, which are then run under Data Logger. 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 this would not be much of a framework if for every module there would need 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 file. This configuration file has a



specific block of configuration settings for every module present in current implementation of Data Logger.[Picture needed]

#### **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 begin to accessing modules based 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 data flow and by which assamble a data handling application in no time. If we would scale this framework to enourmous proportions, 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 id matching of modules. So for every module theres a specific setting to be set, a type, by which we can more easily define routes, e.g. `accelerometer` -> `database` and from now on, every data produced by module which has type set to `accelerometer` will be routed to every module which as type of `database`, may there be big number of them.[Picture needed]

### **3.4.3 Data Logger modules**

In this section of bachelor thesis I will describe modules which I developed in order to complete assignment.

#### **3.4.3.1 Blank module**

During developing modules for this bachelor thesis, I found myself using the same starting point for every module, so I created blank module which has all the necessary functions, or one might call it interface to make more easier creation of new modules, code is commented to give the reader necessary information what is expected from different function in case their name would not be clarifying enough. Blank module combines source and sink together but they do not need to be used both.

#### **3.4.3.2 Time module**

This is simple 'dumb' module which will be used to beta test others modules. Advantage of having this simplistic module is it is not dependent on any hardware. Interval defined by sample rate will provide current time as data.

#### **3.4.3.3 Accelerometer module**

This module will be dependent on Adafruit 10 DOF sensor board and will utilize onboard LSM303 accelerometer sensor, communicating over I2C protocol. Module will emit proper acceleration in 3 axis, maximum acceleration  $\pm 16g$  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, these constants for these settings are implemented, but I do not see the benefit 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 rate, which to my best knowledge is not configurable, 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 Accident module**

This module will be both accepting input data and creating new information for other modules. New cars electronic crash sensors 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 for giving the gathered information value, I am not suitable for the task for simple reason, I am application computer science undergrad and I can create, gather, store, act upon, etc based on information but to assign which value from accelerometer is right to register accident I have no knowledge, it is probably better task for someone from physics field. To create this module smart, 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 centre. 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.

#### **3.4.3.6 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. Modul requires modem to be pin unlocked otherwise SMS sending will fail. Sending SMS message with AT+CGMS command requires string to be sent be a maximal size of 150 characters and ended with ^Z special character. After message is send, module repsonds with OK and number of total SMS messages sent by the module.

#### **3.4.3.7 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, I successfully tested this module on 5 different cars in make and model, but I 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. After successful initialization, obd module will poll suported the car for supported commands, creates a polling loop with them and periodically queries 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 treshold OBD module will not add more commands to queue until the length lowers than treshold. 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.

#### **3.4.3.8 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 modul. 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 form 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.

#### **3.4.3.9 Redis module**

Redis module is database module for storing data in Redis database. As I mention above in Redis overview[TODO], 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.

#### **3.4.3.10 RPM 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 is a treshold under which RPMs drops, module will generate data to indicate to shift gears, similiar action happens with upShiftTreshold, but instead dropping RPMs, it watchs when revolutions are higher then treshold. This module have also configured IFTTT event. This event triggers a notification on mobile device which has IFTTT application installed. In the screenshots below we can see what is looks like[TODO]

#### **3.4.3.11 RabbitMQ module**

RabbitMQ module is primary module for getting data from Rapsberry 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 evaulate 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.

#### **3.4.3.12 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 enviroment but is very handy for debuggind purposes, when modules do not cooperate in expected

way. Console header can be set only to show configuration capabilities in early stages of development.

#### **3.4.3.13 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 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.

### **3.4.5 Main application**

Main application is the core of output program, binds together every feature provided by the solution, and it is the 'brains' for every operation. Application is divisible by features which it offers. In upcoming sections I will describe every each one separately to provide more detail.

#### **3.4.5.1 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 him 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 at GitHub. Server supports websockets to communicate real-time data over them, also for configuring module settings.

#### **3.4.5.2 Web Socket**

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

application as websockets assumes that configuration is wirtten correctly.

#### **3.4.5.3 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 top navigation bar, you can switch between each module configuration, 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 applicaton. Module settings can be changed by clickng configure button. After this you will be presented by form which will allow to add,delete and configure settings. After finishing you can either confirm or cancel new settings. When confirmed, web application will check for syntax correctness. Not acting upon new configruation if it has errors or websocket will be used for sendning 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. For security reasons web application will be password protected as it can be found on costumer-grade routers.

## 4. Implementation

In this chapter of bachelor thesis I will provide a technical implementation details regarding my solution.

### 4.1 Overview

For completing this bachelor assignment I have choosen to implement it in Javascript programming language, which provides various benefits for its implementation.

- Over 70% of the code is implemented in JavaScript, rest is HTML and CSS.
- Application is operating system independent
- For backend solution I 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.

As you can see there is a lot of support surrounding Node.js enviroment which enabled me to build better and faster application.

### 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 configurate 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

user. As this framework is primarily used for fellow computer scientists and power users, Data Logger does not implement any failsafe for not overwriting modules with same ID. This happened to be a mistake to forego 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, whichs value is reffering 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.[TODO TODO TODOTODOTODOOOO TODODO TABLE]

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

### **4.2.2.1 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 EventEmitter 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 by the Data Logger module must emit 'data' event with argument is created data by the module. Nothing more is mandatory to be implemented.

### **4.2.2.2 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.



## **5. Conclusion**

Cieľom diplomovej práce bolo...

V práci som.....

Ďalší možný rozvoj....

# Glossary

**GPIO** General-purpose input/output. 15

**I<sup>2</sup>C** Single-board computer. 15

**IoT** Internet of Things. 13, 14

**OBD-II** On-Board Diagnostics port. 12

**SBC** Single Board Computer. 14, 15

# References

- [1] 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.
- [2] ALMEIDA, Tatiana De. . *About Automatic*. [online]. : <https://www.automatic.com>. [visited on 20.4.2016]. URL: <https://www.automatic.com/press/>.
- [3] AMIT.V.KACHAVIMATH, Nagaraj.C, “Vehicle accident detection and reporting system using accelerometer”, *JII SRD*, vol. 3, no. 4, 2015, ISSN: 2321-0613.
- [4] 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.
- [5] AUTOMATIC. . *How Automatic works*. [online]. : <https://www.automatic.com>. [visited on 20.4.2016]. URL: <https://www.automatic.com/how-automatic-works/>.
- [6] DR. OVIDIU VERMESAN, Dr. Peter Friess 2013. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems* : River Publishers, 2013. . ISBN: 9788792982964.
- [7] EINSTEIN, Albert 1920. *Relativity: The Special and General Theory* : New York: Henry Holt, 1920. . ISBN: 1587340925.
- [8] “News briefs”, *Computer*, vol. 47, no. 7, s. 16–21, Jul. 2014, ISSN: 0018-9162. DOI: 10.1109/MC.2014.189.
- [9] PENROSE, Roger 2014. *The Principle of Equivalence* : Knopf, 2014. . ISBN: 0470085789.
- [10] 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>.

- [11] 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.
- [12] 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](http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=46045).
- [13] . . *VI Monitor Features*. [online]. : . [visited on 20.4.2016]. URL: <http://www.vi-performance.com/?q=node/2>.

# Attachments

## 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

## 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 reset.
- Using a Built-in Virtual Dynamometer, VI can test your vehicles true net horsepower in real world conditions.

## 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[3.4.3.11]
- **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 webserver middleware.
- **gps** – GPS.js is an extensible parser for NMEA sentences, used by GPS module to parse incoming data.[3.4.3.4]
- **i2c-bus** – I2C serial bus access used in accelerometer module[3.4.3.3]
- **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[3.4.3.13]
- **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[3.4.3.9]
- **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[3.4.3.8]
- **serialport** – Node.js library to access serial ports, used in every module which required access to serial port.
- **socket.io** – node.js realtime framework server, used to communicate through websockets or jsonp long polling
- **swig** – A simple, powerful, and extendable JavaScript Template Engine. Used in html templates for web application.[3.4.5.3][<http://paularmstrong.github.io/swig/docs/>]

## Resources

CD obsahující:

- Elektronickú verziu
- Zdrojáky
- atd'