

Saunaxio

Sauna monitoring and remote-control platform

Lauri Korte

Bachelor's thesis

May 2018

Technology, Communication and Transport
Degree Programme in Software Engineering

Author(s) Korte, Lauri	Type of publication Bachelor's thesis	Date May 2018
	Number of pages 128	Language of publication: English
		Permission for web publication: x
Title of publication Saunaxio Sauna monitoring and remote-control platform		
Degree programme Software Engineering		
Supervisor(s) Luostarinen Hannu, Hämäläinen Raija		
Assigned by Rintamäki Marko "NarsuMan"		
<p>Description</p> <p>The assignment came from a private person. The assignor needed a monitoring and remote-control solution for sauna environment, which he had built in his leisure time. The said sauna is an independent structure fitted with wheels, allowing movement with a normal agricultural tractor. The requirement specification was made together with the assignor, along with clear use cases for the system.</p> <p>The project was started by mapping out the existing solutions in general and researching the solutions specifically made for sauna environment in order to create a better understanding of possible and usable technologies and possible tools for the system. The careful research allowed the construction of a whole and functioning solution under the requirement specification and use cases defined for the system.</p> <p>The selection of tools was confirmed at the beginning of the project. The IoT nature of the system was built with Raspberry Pi hardware, allowing independent operating in sauna environment with Node-RED and Home Assistant tools. The latter operated as integration point for the speech recognition system Snips AI allowing the use of voice when operating the connected components, such as applying water onto the stove or operating the music player whilst inside the sauna. The ambient surroundings for the sauna were provided by Mopidy's music player and Festival's TTS service. The data gathered from the local IoT device was sent to cloud infrastructure for monitoring and analys, while the data transfer between the local services was handled by MQTT protocol.</p> <p>The assignment was finished and it met the requirements and use case specifications in total. The modifications to the initial design were executed with reliability and usability in mind. The finished project acted as a demo, which later was deployed to the final placement in sauna environment.</p>		
Keywords (subjects) IoT, sauna, Node-RED, Home Assistant, Snips AI, monitoring, TTS		
Miscellaneous		

Tekijä(t) Korte, Lauri	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Toukokuu 2018
	Sivumäärä 128	Julkaisun kieli Englanti
		Verkojulkaisulupa myönnetty: x
Työn nimi Saunaxio Saunan monitorointi ja etähallinta alusta		
Tutkinto-ohjelma Ohjelmistotekniikan koulutusohjelma		
Työn ohjaaja(t) Luostarinen Hannu, Hämäläinen Raija		
Toimeksiantaja(t) Rintamäki Marko "NarsuMan"		
<p>Tiivistelmä</p> <p>Toimeksiantajana toimi yksityishenkilö. Toimeksiantajalla oli tarve toteuttaa monitorointi- ja etähallinta ratkaisu saunaympäristöön. Kyseinen sauna oli toimeksiantajan vapaa-ajallaan rakentama liikuteltava kokonaisuus, jota tarvittaessa voitaisiin siirtää erilaisille tapahtuma-alueille. Tarkastelun kohteena oleva sauna on itsenäinen pyörillä varustettu rakennus, jota voidaan liikuttaa normaalin maataloustraktorin voimin. Vaatimusmäärittely rakennettiin yhdessä toimeksiantajan kanssa, jolloin muodostuivat myös selkeät käyttötapaukset toteutettavalle työlle.</p> <p>Pohjatyö aloitettiin kartoittamalla olemassa olevia ratkaisuja sekä yleisesti että nimenomaan saunaympäristöön toteutettuja sekä näiden pohjalta mahdollisia käytettäviä teknologioita. Työ suunniteltiin valitsemalla mahdollisia työkaluja vaatimusmäärittelyn ja käyttötapauksien valossa; työkalujen yhteyksiä tutkittiin, jotta voitaisiin saavuttaa eheä kokonaisuus.</p> <p>Toteutuksen alkuvaiheessa valikoituivat varmistuneet työkalut, joilla työtä lähdettiin toteuttamaan. Kokonaisuuden IoT-luontainen olemus rakennettiin Raspberry Pi -laitteistolla, jonka itsenäinen toiminta varmistettiin Node-RED ja Home Assistant -ympäristöillä. Jälkimmäinen työkalu toimi myös liitoskohtana puhetunnistukseen erikoistuneelle Snips AI -palvelulle, joka mahdollisti äänen käytön komponenttien hallinnassa saunaympäristössä, kuten löylyveden annostelussa ja musiikkisoittimen käytössä. Saunaympäristön ambienttinen puoli toteutettiin Mopidy-musiikkisoittimella sekä TTS- teknologiaa hyödyntävällä Festival-ohjelmistolla. Datan keruu IoT-laitteelta ohjattiin pilvipalveluun datan monitorointia ja analysointia varten, paikallisella laitteella työkalujen välisestä kommunikaatiosta vastasi MQTT-protokolla.</p> <p>Toimeksianto saatiin päätökseen vaatimus- ja käyttötapausmäärittelyiden täytyttyä kokonaisuudessaan. Muutokset alkuperäiseen suunnitelmaan toteutettiin toimintavarmuutta ja käyttäjäystävällisyyttä silmällä pitäen. Työ toteutettiin demona pöytäympäristöön siirrettäväksi myöhemmin varsinaiseen toimintaympäristöön.</p>		
Avainsanat (asiasanat) IoT, sauna, Node-RED, Home Assistant, Snips AI, monitorointi, TTS		
Muut tiedot		

Contents

Terminology	7
1 Introduction	9
1.1 Overview and client.....	9
1.2 Thesis goals	10
2 About the solution	12
2.1 Designing the solution.....	12
2.2 Existing solutions	12
3 Technologies	13
3.1 Internet of Things.....	14
3.1.1 Background.....	14
3.1.2 Basic structure.....	15
3.1.3 Address pool size.....	15
3.1.4 IoT security	16
3.2 Cloud Computing.....	17
3.2.1 Cloud types.....	17
3.2.2 IaaS, PaaS & SaaS	18
3.3 Automation	19
3.3.1 Background.....	19
3.3.2 Home automation.....	20
4 Sauna.....	21
4.1 Background.....	21
4.2 Operating principle.....	22
4.3 As an environment.....	23
5 Tools in development.....	24
5.1 Home Assistant	25
5.2 Node-RED	26
5.3 Snips	28
5.4 Mosquitto and MQTT	29

5.5	Mopidy.....	29
5.6	Festival TTS	31
5.7	DigitalOcean	31
5.7.1	InfluxDB.....	32
5.7.2	Grafana.....	33
6	Hardware.....	34
6.1	Raspberry Pi.....	34
6.2	Sensors and components.....	35
6.2.1	Temperature sensors.....	37
6.2.2	Humidity sensors.....	41
6.2.3	RuuviTag.....	42
6.2.4	Ultrasonic sensor.....	43
6.3	Sound	44
6.3.1	Speaker	44
6.3.2	Microphone.....	45
6.4	Fieldbox.....	46
7	Building the solution.....	50
7.1	Software architecture	50
7.2	Raspberry Pi.....	53
7.2.1	Installation and backup.....	53
7.2.2	Internet connection	54
7.2.3	Configuration	56
7.3	Connecting the hardware.....	58
7.3.1	Sensors.....	59
7.3.2	Other hardware	61
7.4	Node-RED	62
7.4.1	Upgrading Node-RED.....	62
7.4.2	Installing required nodes	63
7.4.3	Configuring Node-RED	66
7.4.4	Building flows	68
7.5	Mosquitto	74
7.6	Home Assistant	76

7.6.1	Configuration	77
7.6.2	Adding components.....	78
7.6.3	Customisation	82
7.6.4	Automation	86
7.7	Cloud services.....	88
7.7.1	Influx database	88
7.7.2	Grafana install and configuration.....	91
7.7.3	Adding data source to Grafana	94
7.8	Audio system.....	96
7.8.1	Connecting Bluetooth speaker	96
7.8.2	Audio output.....	97
7.8.3	Audio input.....	99
7.8.4	Text-To-Speech.....	100
7.9	Mopidy.....	103
7.10	Snips AI.....	107
7.10.1	Configuration	107
7.10.2	Assistant.....	109
7.10.3	Using Snips AI	112
7.10.4	Assistant installation problem	113
8	Evaluation	115
9	Future development.....	116
10	Conclusion.....	117
	References.....	119
	Appendices.....	123
Appendix 1.	Block diagram of the system	123
Appendix 2.	Fieldbox layout with circuit board	124
Appendix 3.	Sequence diagram of Snips AI function.....	125
Appendix 4.	Hardware list.....	126
Appendix 5.	Hardware placement inside the sauna.....	127
Appendix 6.	QR-code for the Github repository (URL included).....	128

Figures

Figure 1. Sauna environment.....	9
Figure 2. Idea of the Internet of Things.....	15
Figure 3. Pizza as a Service (Barron 2014)	18
Figure 4. Example of a smoke sauna (Smoke sauna 2004)	22
Figure 5. Example of Home Assistant overview	26
Figure 6. Flow example for RuuviTag in Node-RED.....	27
Figure 7. Node-RED Dashboard view.....	27
Figure 8. View from Snips Console	28
Figure 9. Mopidy Musicbox Webclient user interface	30
Figure 10. InfluxDB command line interface	32
Figure 11. Dashboard view of RuuviTags in Grafana.....	33
Figure 12. Raspberry Pi 3 Model B used in the solution.....	34
Figure 13. Raspberry Pi specifications for different models (Benchhoff 2016)	35
Figure 14. Example of sensor casing	37
Figure 15. Both versions of the DS18B20 sensor	38
Figure 16. Thermocouple K-type with MAX6675 converter	40
Figure 17. DHT22 temperature and humidity sensor	41
Figure 18. RuuviTag with waterproof casing.....	42
Figure 19. Waterproof ultrasonic sensor AJ-SR04M.....	44
Figure 20. UE Roll 2 Bluetooth speaker	45
Figure 21. USB microphone used in solution.....	46
Figure 22. Fieldbox before installations with battery connected	47
Figure 23. 8 Relay Module used in solution.....	48
Figure 24. DC/DC converter used in solution.....	49
Figure 25. Circuit board constructed for the setup	50

Figure 26. Flowchart of the system	51
Figure 27. Raspberry Pi 3 GPIO pins as seen from gpio.....	57
Figure 28. Raspberry Pi 3 GPIO layout (Raspberry Pi 3 Pinout 2016)	59
Figure 29. Flow to handle RuuviTag data.....	68
Figure 30. Inject and trigger nodes.....	69
Figure 31. Message from the RuuviTag node.....	70
Figure 32. Function node for message modification.....	70
Figure 33. Switch used to sort out RuuviTags	71
Figure 34. Creating InfluxDB node connection	72
Figure 35. Node-RED's MQTT broker node	73
Figure 36. Exec node for system commands.....	74
Figure 37. <i>Time&Date</i> component in Home Assistant.....	79
Figure 38. Executable scripts in Home Assistant	82
Figure 39. Group view of Home Assistant.....	83
Figure 40. Customisation view of Home Assistant	84
Figure 41. Custom UI showing <i>last_active</i> in the Home Assistant	85
Figure 42. Trigger for automation.....	87
Figure 43. Action for automation.....	87
Figure 44. Measurement for RuuviTags.....	91
Figure 45. Adding a notification channel for Grafana.....	93
Figure 46. Alert rules for RuuviTags	94
Figure 47. Creating graph for RuuviTags	95
Figure 48. Graph for RuuviTag temperatures	95
Figure 49. <i>D3 Gauges</i> used for RuuviTag singlestat	96
Figure 50. Mopidy-Moped UI	105
Figure 51. Mopidy-Mopify UI view.....	106
Figure 52. Creating skills in Snips Console	110

Figure 53. Training the skill with training examples	111
Figure 54. Query for the new skill	112
Figure 55. Snips speech recognition flow	113
Figure 56. Message shown after download fails.....	114

Tables

Table 1. Functional requirements	11
Table 2. Used components and parts	35
Table 3. Additional Node-RED node installations.....	64

Terminology

ALSA	Advanced Linux Sound Architecture. An API for Linux kernel's sound card device drivers.
API	Application Programming Interface. A method allowing different software to communicate with each other.
ASR	Automatic Speech Recognition. Speech recognition that allows the use of normal human language to control a computer device.
BLE	Bluetooth low energy. A short range Bluetooth technology used by many wireless devices, such as RuuviTag beacon.
GPIO	General Purpose Input/Output. A general purpose pin found on computer boards, such as Raspberry Pi.
IoT	Internet of Things. A technology, where appliances may be connected together and to the Internet, gathering data or functioning for some advantage for the user. Connected devices are recognised with identifiers, such as an IP address.
JACK	JACK Audio Connection Kit. A sound server daemon for audio data using its API, supporting the ALSA driver.
MPD	Music Player Daemon. An open-source music player software, with player and user interface separated from each other. Possibility for user interface with client softwares.
MQTT	Message Queuing Telemetry Transport. A light-weighted publish-subscribe messaging protocol requiring a message broker to function. Designed to work with limited bandwidth.
NLU	Natural Language Understanding. An engine for parsing a query in text format used by Snips AI.

TLS/SSL	Transport Layer Security, formerly Secured Sockets Layer. An encryption protocol for secure communication over network.
TTS	Text-to-Speech. A technology, where normal human language text is turned into synthesised speech.
UI/GUI	User interface and graphical user interface.

1 Introduction

1.1 Overview and client

The market for the Internet of Things has been vastly increasing, with the basic idea of different devices or appliances, the ‘things’ in the IoT world, communicating together and gathering data for some advantage. The growing demand open numerous possibilities for many amateur projects, whether it be a small, personal weather station or something more complex, such as a fully automated and controllable home. Whatever the implementation may be, the estimated number of ‘things’ connected in a way of the Internet of Things, is well over 11 billion by the end of year 2018, according to Gartner. (Gartner Newsroom 2017)

The traditional Finnish sauna is an old one and not much technology can be found within. The conventional setup can be broken, as it will be in this Bachelor’s thesis work. Many are the people who may object bringing technology inside the sauna, and many rightfully so. The aim is not to turn the basic sauna into full-on technology centre, but to implement an IoT solution in a way it does not affect the conventional nature of the sauna (Figure 1).



Figure 1. Sauna environment

The thesis containing the said solution was assigned by the writer's former teacher Marko Rintamäki at JAMK University of Applied Sciences. The client had a vision of an IoT solution implemented into sauna environment, and in this case, into a mobile sauna. This mobile sauna, built on a trailer, was well suitable as a development environment for the IoT solution. The writer was happy to accept offered work and together with the client the basic background knowledge of said solution was easily acquired.

1.2 Thesis goals

Thesis work had a main goal of developing a functioning solution for a sauna monitoring and remote-control system and building the platform for this purpose. The monitoring was to be handled by data gathering from sensors on the local machine and presenting the data in a cloud service, as well as on the local machine for more current data, while the monitoring service in the cloud would show change for longer period of time.

The sauna had to be functional with and without the central core of operations and its services and the technology inside the sauna should not be exceedingly visible to the bather inside. The sauna structure in whole, including the technological parts, was to withstand not only the sauna environment but also the surrounding environment of the structure, given the characteristics of the said structure being external from a house with no base heating. The system would continue to operate off battery, whenever the mains would be disconnected.

The hardware inside the sauna structure, including the ones in the fieldbox, was to be removable or replaceable at leisure due to desire or error occurrence within the devices. The bather should be able to control the devices, with either browser or mobile application, when outside the sauna, whilst inside the controlling should be done with speech recognition service.

Use case specification and requirement specification were done with the client at the start of the project, with modification made during the development. The

initial use cases are listed below and referenced in thesis in corresponding places, where the said use case or requirement was addressed.

Table 1. Functional requirements

<i>Req. #</i>	<i>Description</i>
<i>RE100</i>	System operates independently, limitedly without Internet connection
<i>RE200</i>	Steam amount can be adjusted manually with a switch
<i>RE201</i>	Possibility for automated steam adjustment
<i>RE210</i>	Steam adjustment with a voice command
<i>RE300</i>	User may enquire current temperature with a voice command
<i>RE301</i>	User may enquire current time of day with a voice command
<i>RE302</i>	User may enquire current relative humidity with a voice command
<i>RE400</i>	User may control lights
<i>RE410</i>	User may control lights with a voice command
<i>RE500</i>	User may control music player
<i>RE501</i>	User may select playlist for music player
<i>RE510</i>	User may change track with a voice command
<i>RE511</i>	User may play and pause the track with a voice command
<i>RE512</i>	User may change player volume with a voice command

2 About the solution

2.1 Designing the solution

The initial design of the solution was done during the Autumn of 2017. The designing consisted of researching different possibilities of making the solution described, finding the correct platforms and tools for cloud services and for the hardware. Many IoT projects with similarities of the project at hand was found in different sources. Some of the tools to be used, such as the Node-RED and Home Assistant, as well as the cloud service provider and the platform to build the IoT project on, the Raspberry Pi 3, were predefined, as they were preferred by the client.

The initial design of an IoT solution was as described earlier. An independently operating system inside the sauna, with little to none technology visible for the bather inside, who could operate the specified functions with either own voice commands or remotely with mobile application or from a browser. The data sent to the database could be monitored by the user and shown on a screen outside the sauna, on the outer wall.

The specifications provided a good platform from where to start building and developing the final solution. The potential additions or changes made during the building and developing of the system are described in the actual building part of thesis work (See Chapter 7).

2.2 Existing solutions

Many similar IoT projects, about monitoring, automation and remote controlling, can be found around the Internet as simple or more complex projects, some even very close to the setup built in thesis. Below are three easily found examples of similar versions concerning the IoT projects built in a sauna environment.

One solution found monitors the temperature of a sauna or steam room and sends notifications for the bather about temperature readings to notify the bather

the room temperature is at preferred level. Similarly, the notifications are sent if the temperature exceeds the set levels, at which point the heater could be turned off (Internet Sauna & Steam Room Alert 2016.)

Another solution offers closer similarities than the ones specified for the thesis. While this solution has no automation available, the temperature is monitored and shown on graphs to the user. The solution uses the DS18B20 temperature sensor, fitted inside the wall panelling. Notifications are sent based on the current temperature, when the sauna is ready, heated to a proper level, or when the temperature exceeds a certain point (Marjamaa 2016.)

Third solution is closest to the one in thesis work, considering the requirements. In this IoT solution the temperatures are monitored from different sources, from inside the sauna, from inside and outside of the house and the current water temperature of a nearby lake. The Home Assistant is used to operate appliances with the Z-Wave wireless communications protocol. As the core of system both Raspberry Pi and Arduino are used for different locations, Arduino being used where Raspberry Pi's size or power is excessive. The communication between the devices is handled with MQTT protocol (Mäkinen 2016.)

While IoT projects are mostly designed for normal environment, either indoors or outdoors, the examples above were all designed for the sauna's environment, with the first of said examples designed to function more in a business manner, while the other two examples were projects for more private surroundings.

3 Technologies

As the device to be inside the sauna reads the sensors and sends the data to be monitored in a cloud service, as well as the device having the possibility of having automated functions, and some remote controlled in that matter, describing what the technologies are is essential for the understanding of the solution. The main technologies are opened in the chapters following. The environment for the solution, the sauna, is described in its own chapter.

3.1 Internet of Things

3.1.1 Background

Internet of Things, or IoT for short, is, as a term, still quite young. According to writing in Postscapes the term was first used by a man named Kevin Ashton in one of the presentations he held in 1999. Although the basic idea is that the devices, the 'things', communicate together over the Internet and gather data for some purpose, the described setup is older than the term and the Internet (IoT History 2016.)

Approximately 200 years ago, in the second quarter of the 19th century, the first fully functional electromagnetic telegraph was invented and for the first time in history two devices could communicate together with the use of electricity, which in return was quite new technology as well. However, this communication between the telegraphs was made possible by the machine users interacting with them directly, which is some way of from the Internet of Things idea; devices being able to communicate and function separately and individually, yet in whole, without the potential interference of a human factor (IoT History 2016.)

In 1926, a man named Nikola Tesla gave an interview for the Colliers magazine, stating that "*When wireless is perfectly applied the whole earth will be converted into a huge brain, which in fact it is, all things being particles of a real and rhythmic whole...*". Although this statement considered the wireless telegraph and not the Internet, the idea itself was futuristic and, when thinking about today's world and all the IoT devices and implementations, even surprisingly quite accurate (IoT History 2016.)

In present day this old idea has been applied into many different appliances and purposes. With the global Internet access many of these devices can function on their own and the collected data can be accessed by the user from basically anywhere on the planet.

over 4,2 billion because of the 32-bit system it uses. For normal use, computers and such, this is still adequate, however, from IoT's perspective it is quite small.

With the IPv6 and its 128-bit system, the number of addresses immensely go up. With unlimited amount of possible addresses to be given for different devices of IoT nature, the world can be filled with different implementations, creating unlimited possibilities of monitoring the living or the surroundings for some advantage (Rouse 2016.)

3.1.4 IoT security

With said possibilities of unlimited IoT implementations in the world one cannot ignore the importance of the security of said implementations. The basic knowledge of, for example software development, is to design the software or system to be secure from the start, not to implement the security part later on when it is already too late or extremely hard to implement correctly. This can be tedious but, especially for IoT devices that can monitor or control the user and one's life, also very important.

One basic security matter is to have secure authentications and correct authorisations for the device itself and the network, where said IoT devices operate. There is a difference between authentication and authorisation. The former is to identify the user, who is using or accessing the devices. The latter is to give certain permissions of what one can do while using or accessing the IoT device. With a proper implementation of correct safety features the possibility of third-party member gaining access to the devices are minimised (Gite 2012.)

The same setup may be used once the IoT implementation is potentially outside the secured local area network, for example in the mobile network. While the mobile network is not secured as the local area network would be, a greater thought is to be laid on the IoT device itself.

As part of the IoT nature, the IoT device must be able to send the gathered data safely, without the third-party member intervening with it. With the secured IoT

device itself and the connection between the device and the database, all that remains is to secure the database in the cloud (See Chapter 7.7.1).

3.2 Cloud Computing

3.2.1 Cloud types

On the surface the cloud computing is usually divided into three groups; public, private and hybrid clouds. Each of these types has its own strengths and weaknesses.

The most used cloud type is probably the public cloud. In public cloud the service provider is responsible for making the resources, for example applications and storage, available for a large mass on the internet. This type can be free, at least to some point, or it can be chargeable by the use (What is cloud computing.)

The second type, the private cloud, has the same benefits as the public cloud, the scalability being one of them, but instead of it being available to basically the whole crowd on the internet, the private cloud is only available to either one person or a single organisation. The private cloud is best used for organisations who want to be in control of what is happening in the cloud. Because of this the security matter can also be addressed more rapidly than in a public cloud. The cost for private cloud is higher, since the organisation usually maintains and updates all the needed resources by themselves (What is cloud computing.)

The third type is the hybrid cloud. As the name implies, the hybrid cloud is a combination of the first two cloud types. The combination grants the possibility to use the public cloud for less important or sensitive data while using the private setup to store the important and more sensitive information, without paying any more that is necessary (What is cloud computing.)

3.2.2 IaaS, PaaS & SaaS

When examining the lower levels and meaning of cloud computing, different models, the Infrastructure as a Service, Platform as a Service and Software as a Service, can be found, each having their own characteristics. To simplify the terms, see the picture (Figure 3).

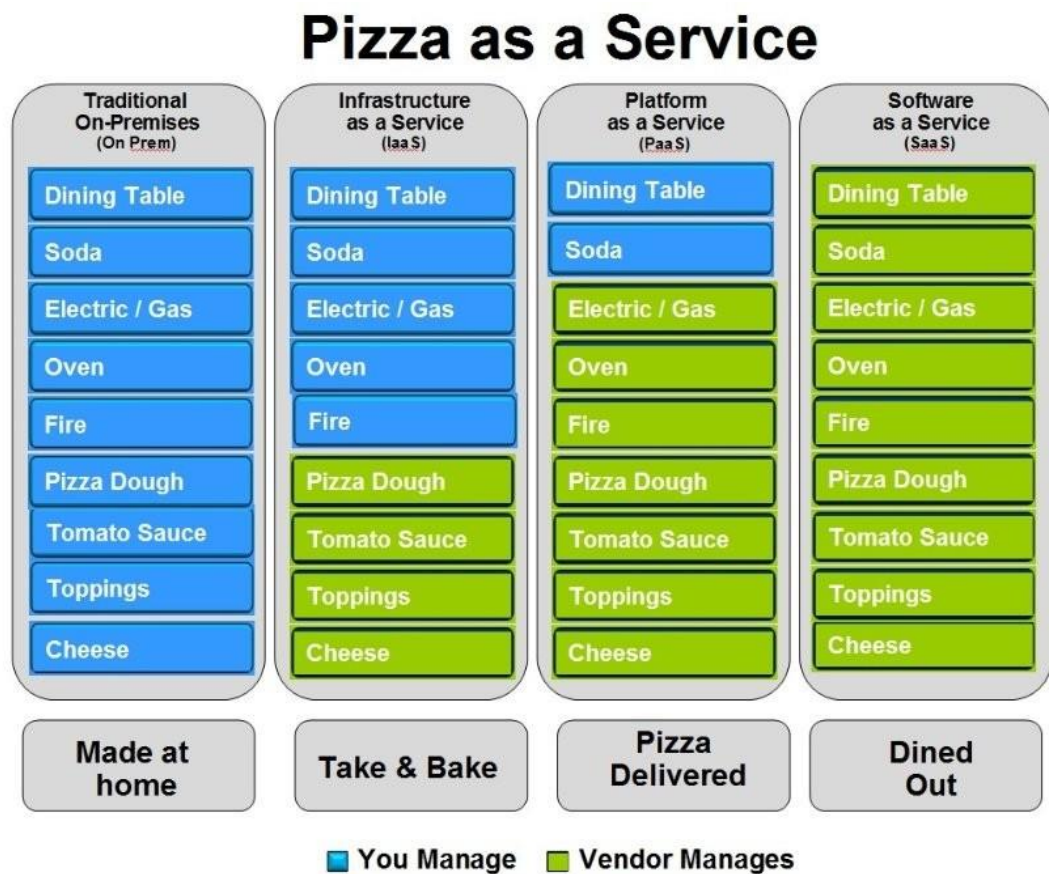


Figure 3. Pizza as a Service (Barron 2014)

The traditional and old version of the cloud service, for example a server machine, is to have the machine on personal possession (far left on figure above). This allows full control over the services running on said machine but also leaves the responsibility of its maintenance and safety to the user. As opposed to this method, the IaaS, PaaS and SaaS offer different approaches when building the cloud services.

The first one, the Infrastructure as a Service, offers virtualised computing resources for the user in a safe environment. As the security and maintenance is handled by the service provider the user can concentrate on building and

developing their own IT platforms. The charge for the service is low and usually based on the use and the service is highly scalable, should there be need for more space or power. An example of IaaS, to be used in thesis work as well, is the DigitalOcean, providing the cloud service for the database to store the future data (What is IaaS.)

The second model is the Platform as a Service and it lands on top of the IaaS. In addition to the service provided by the IaaS service provider, the environment on which to build applications and share them over the Internet, is also provided by the service provider. The PaaS can provide operating systems, server software, storage, design and development tools and overall support for the user. Such services used are for example the Heroku that user can use to run and develop their own software and make it available for others to use (What is PaaS.)

The final model is the Software as a Service and is the simplest model for the user. With this model the user rents or buys the software itself, which is on top of the PaaS and IaaS. This is the most used one and many use the SaaS without even knowing it's there. Users subscribe to the software and are charged on monthly basis or by the use. Some of the biggest used software are, for example, the Netflix, Office 365 and Google Apps (What is SaaS.)

3.3 Automation

3.3.1 Background

Automation is a large concept that has been part of the world for centuries. The basic idea of automation is to have a process to accomplish on its own, without needed interaction of human nature after the initial setup of the automation process.

One of the first 'automated' contraptions were simple steam powered devices developed by the ancient Greeks but served little to none purpose. In the Middle Ages first mechanical clocks started to arrive to Europe, with a weight to keep the clock operating as an automation (Groover 2018.)

The Industrial Revolution increased the number of steam powered devices. Early steam engines and the pressure inside was handled manually, as the boiler would explode were the pressure to increase above certain limit. This was later automated with a regulator valve that operated based on the current pressure inside the boiler and would decrease the pressure to maintain proper level, ensuring automatically the boiler would not explode and the running speed and power of the steam engine could be managed (Groover 2018.)

3.3.2 Home automation

The home automation is quite young concept, as it has been around for about 80 years. According to Jennifer Tuohy's writing in Network World (2015), the concept was first introduced in Chicago World's Fair in 1934 as 'home of the future'. The idea back then was to connect devices and appliances together into centralised control panel inside the home, where the devices could be used remotely, without touching the device directly.

In 1975 one of the first moves towards larger home automation was made with the introduction of the X10 protocol. With the X10 all devices connected together with electrical wires could be communicated with and controlled. The X10 transmitter's signal consists of numerical code with the information of command being issued, an identification code to the device commanded and finally the command what the device should do. Designed to be fast communication, in reality the channels could get stuck and jammed of several messages and because of the electricity itself in the wires, resulting in false commands for false devices (Edmonds, Chandler.)

The regulator valve described earlier is based on negative feedback, or balancing feedback, a method that is used on many automated devices and processes. Today a common sight is a thermostat to handle the current and desired temperature of a room or home. The thermostat functions with the negative feedback; when the temperature drops below the set level, the switch activates turning on the heaters, increasing the temperature back to desired level and then switching back off. This method is simple and effective automation which can be

pictured as pendulum effect, where, in this case, the temperature decreases and increases relatively smoothly around the desired level (Groover 2018.)

As of the Internet of Things, increasing numbers of different devices and appliances in home environment can be implemented into smart home concept. In smart home concept the previously centralised home automation being controlled from a control panel, is brought to the user's smartphone on an application to handle the devices and appliances from basically anywhere with an Internet connection. This enables the mobility of the services; the home monitoring can be accessed from work and appliances activated and deactivated from outside the home's local area network. Strong security is, as in many cases, essential as previously described in the IoT section of thesis work.

4 Sauna

4.1 Background

Since the term 'sauna' is quite large as it may signify many relatively different things and places, first it is important to know what 'sauna' means in this thesis work. The sauna in this case specifically is a Finnish sauna.

The sauna is believed to have developed from the ancient dwellings, larger holes in the ground covered up with leathers and sprigs. These dwellings were kept warm with fire inside and the best way to reserve the warmth was with stones. The fire was kept under the stones, which could then heat up the space for a long time after the fire had extinguished. The fire was usually close to the door, so that the smoke could easily escape the dwelling.

At some point water was 'invented' to be added on the stones, thus creating steam and sweating, which in addition could erase even the hardest of dirt on a person and increased the cleanliness of inhabitants, as they could easily bathe on benches higher up in the sauna. These three main elements remain till present day; stones to reserve heat, water, or *löyly*, to maintain the warmth feeling and bench, *laude*, to sit and bathe on.

Sauna spread across the European continent and was quite popular as heating source. After more established buildings with fixed fireplaces were erected, saunas lost their meaning as places to bathe. In Northern and Eastern Europe sauna was still considered as an ideal place to bathe and keep warm, as the people used to travel a lot, and no actual buildings were present.

The traditional sauna version was the smoke sauna, *savusauna* (Figure 4), described earlier, till the very end of 19th century, when people slowly started to prefer more modern woodburning stoves, with electric heater version starting to arrive in mid 20th century. The most used versions today are the woodburning and electric heater stoves, with the former being the one used in thesis work setup as well (Saunan kehitystä 1997/1999.)



Figure 4. Example of a smoke sauna (Smoke sauna 2004)

4.2 Operating principle

This chapter concentrates on woodburning sauna and its basic operating principle but applies on many parts to the electric version as well.

The stove, or *kiuas* in Finnish, produces heat to heat up the stones on top of the stove. These stones reserve the heat and are used to control the warmth and feel of the heat inside the sauna. Water, or *löyly*, thrown onto the stones create steam, increasing the humidity of the sauna and making the air feel warmer. The normal relative humidity readings vary between 10 and 30 percent, with the temperature normally staying between 70 and 100 degrees Celsius.

Air circulation is essential for the well-being of the bather and also for the woodburning stove. The supply air pipe is usually located somewhere close to the stove and potentially slightly above the stove. The colder air drops down and starts circulating with the warmer air rising from the stove, giving fresh air for the bathers. If the air supply is located close to the floor, the air should be channelled up, closer to the stove top (Saunan rakentaminen – Tuloilma 1997/1999.)

The exhaust air pipe is usually located at the back of the room, under the benches. As the air circulates the room, air drops down and exits through the exhaust air pipe, giving steadier temperature readings around the sauna room and healthier experience (Saunan rakentaminen – Poistoilma 1997/1999.)

No pre-defined time is present for the bathing session, as the sauna experience is individual. Depending on the person's own likings, the bathing can last from just couple of minutes to hours. Longer periods of time in a sauna is unhealthy, and bather must take regular breaks to cool off and rehydrate.

4.3 As an environment

The sauna is not a conventional environment for an IoT project. As the normal room temperature of $\sim 20^{\circ}\text{C}$ is the 'normal' environment, the components and IoT devices are usually designed for this specific environment as well. While the temperature and humidity increases, a special care is needed when selecting the correct devices and components for the project, in order to ensure good life expectancy and safety of said devices.

Many sensors found for sale in various stores are designed to cope the stress of high temperatures and humidity, at least for the sauna standards. Components are easy to select from specific category and measuring range but more complex devices, such as speakers and microphones are harder to find, from the lower price categories.

The continuous exposure to high temperature and humidity can affect some of the devices and components on such level these devices stop functioning as intended. To prevent such from occurring the more delicate devices have to be located in a secluded area outside the immediate danger zone. As of the mobile nature of the sauna, being built on a trailer, not only is the environment inside the sauna a problem but also the environment outside the structure. The sauna structure will be located outside throughout the year, and during the coldest periods of time in winter, certain components or devices can be removed from the sauna, if not needed, to prevent unnecessary damage or wearing, the more delicate devices being the Raspberry Pi 3, the speaker and the microphone. The latter is quickly removed due to the USB connection to the Raspberry Pi.

The mobile nature of the sauna causes problems for the power of the devices and for the Internet connection. For this reason, the sauna structure is fitted with solar panels to address the power problem, and network router for the Internet address, to which the Raspberry Pi is connected either physically with a cable or with WiFi connection.

5 Tools in development

This chapter concentrates on the major tools and technologies chosen for the development and building of the solution. The used hardware is specified later.

The tools were selected with a few basic requirements. All of the tools were to be easy to install and implement, easy to use and modify, and all of the tools were to be open-source to keep the cost of the project as low as possible, except for the cloud infrastructure provided by DigitalOcean, which was charged monthly. The

pool of tools available for use is large and the selection of the following tools was made with usability and relevance in mind for this specific solution.

To see the architecture and reasons behind the major tools used in the project see the software architecture (See Chapter 7.1).

5.1 Home Assistant

The Home Assistant is an open-source home automation platform that runs on Python 3. Designed to be used on the Raspberry Pi, the Home Assistant can track and control numerous devices located in the same network. The Home Assistant functions on modular bases, as the implementation of new component integrations are made easy. While other automation platforms may need to use the power of cloud services to operate and handle the automations, the Home Assistant runs only on the local device, whether it be a normal PC or Raspberry Pi, with only limiting factor being Python and its compatibility of said devices. Home Assistant has possibility for browser-based (Figure 5) automation and monitoring but can also be accessed through mobile application for more mobility (Brown 2016.)

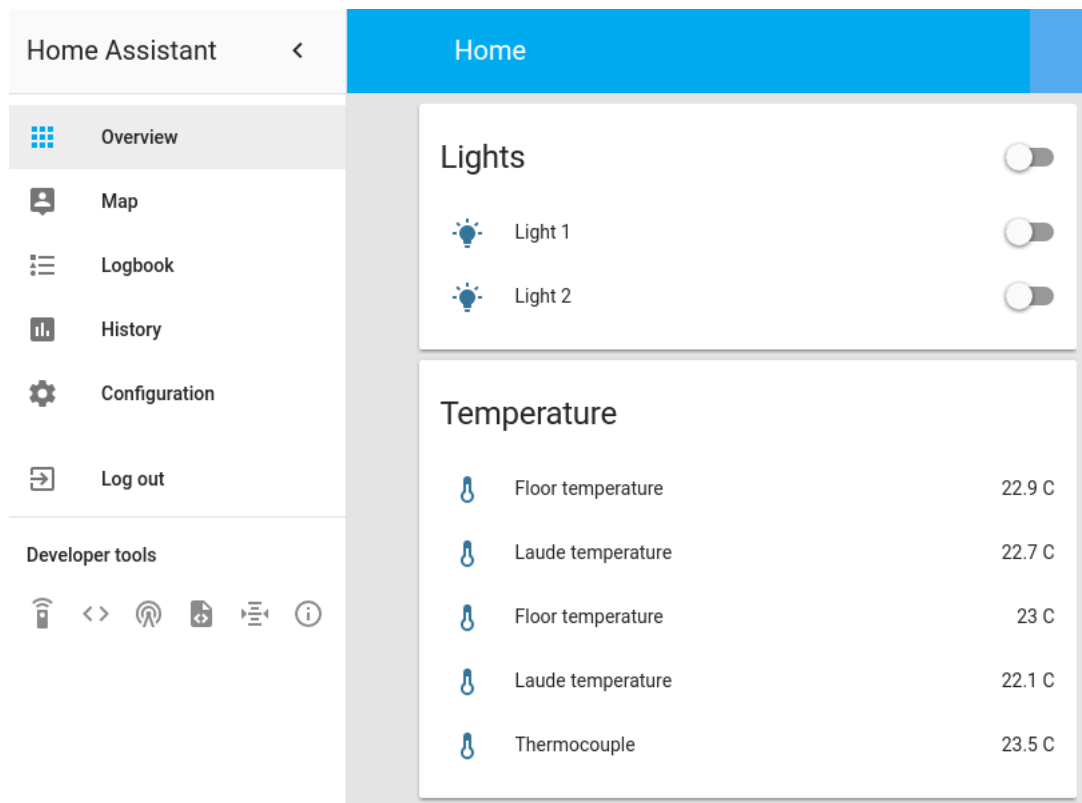


Figure 5. Example of Home Assistant overview

In this thesis work the Home Assistant is used to control and track all available components inside the sauna, from Mopidy MPD music player to lights and water pump. Certain components may be used from Snips AI, a speech recognition system, as well, as the possibility of handling a smartphone in a sauna for a longer period is bleak.

5.2 Node-RED

The Node-RED is a graphical flow-based programming tool to wire up different IoT devices and services. The different modules, whether physical devices, application programming interfaces (API) or online services, can be connected to create complex flows for IoT purposes. Built on the Node.js the Node-RED can be accessed and operated from a browser environment (Figure 6). While probably the most used environment is on a physical device, usually on a Raspberry Pi in that matter, the Node-RED can also be implemented into cloud environment. The Node's package repository contains well over 200,000 modules, which can be added at leisure to further increase the possibilities with the Node-RED. Node-

RED also offers a bunch of UI nodes to implement a simple UI (Figure 7) for easy access to current data (Node-RED 2017.)

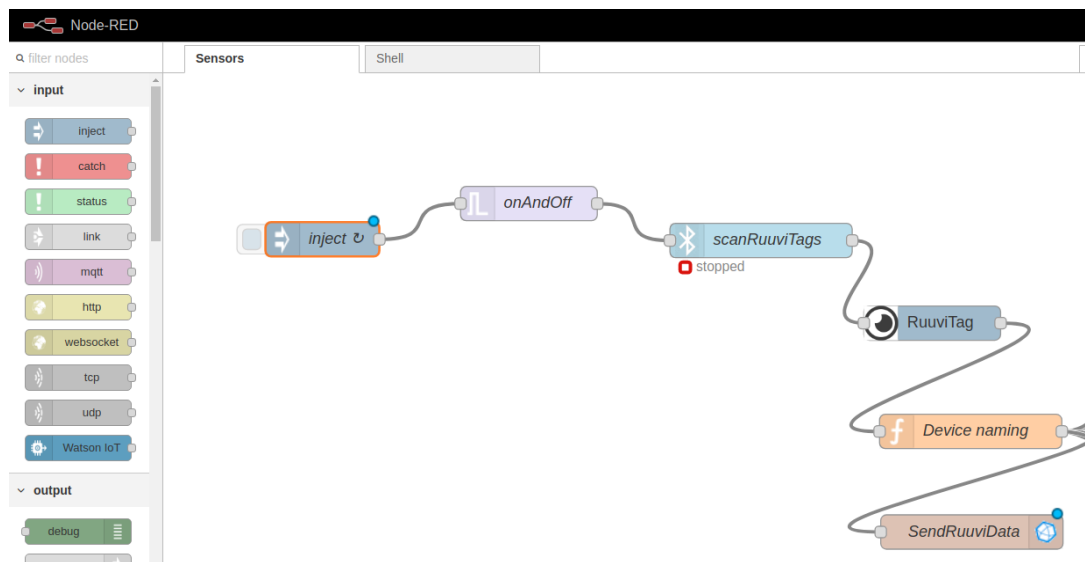


Figure 6. Flow example for RuuviTag in Node-RED

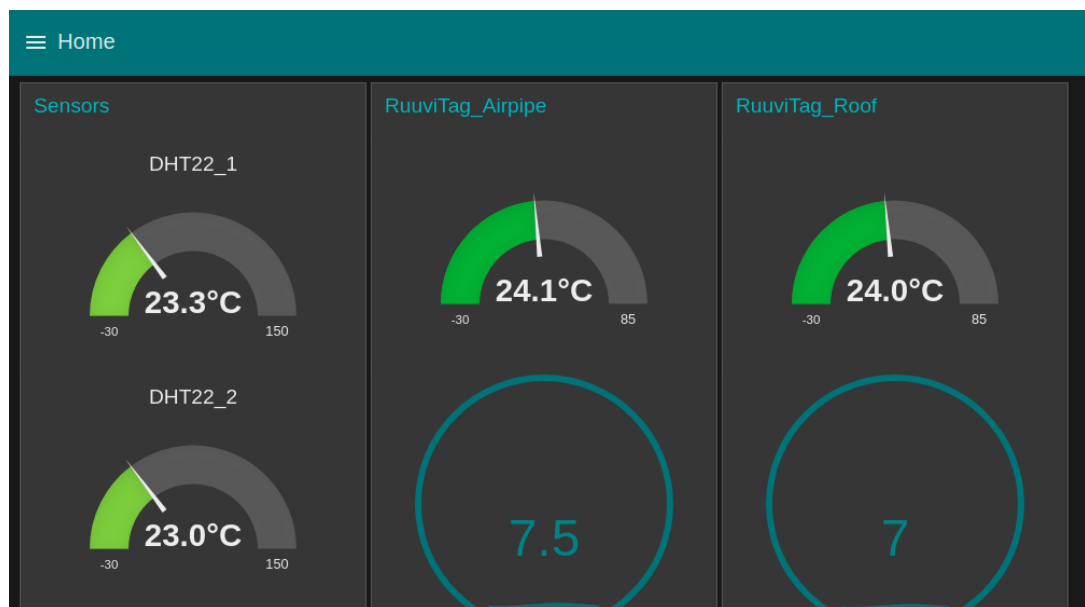


Figure 7. Node-RED Dashboard view

In the solution the Node-RED is used to gather the data from the sensors, modify the data to match preferred template and to send the data to the cloud database. Node-RED also handles the data transferring to Home Assistant with Mosquitto MQTT to show the latest data, without the necessity of getting the data from the database. In addition, some of the shell commands are executed from the Node-RED.

5.3 Snips

Snips Voice Platform allows to have AI powered voice interaction in many different devices. The platform contains Hotword detection, Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU), which all run using deep learning locally on the device, meaning the cloud services are not needed.

The Snips' function is based in bundles, which contains intents that all have their own purpose. The Snips Console can be used to add or modify either readily-made bundles and intents or to make own bundles with own intents to work on (Figure 8). The Assistant is downloaded and installed on the local device, which is used in the speech recognition.

The Snips Platform detects when a hotword, or a wake word, is captured and takes user speech as input. The input is then transcribed, analysed for intents from the installed assistant and finally searched for special slots in the intents. At this point the results can be handled by the user's code as seen fit. The communication between the different Snips services is done with MQTT broker (Snips Voice Platform 2018.)

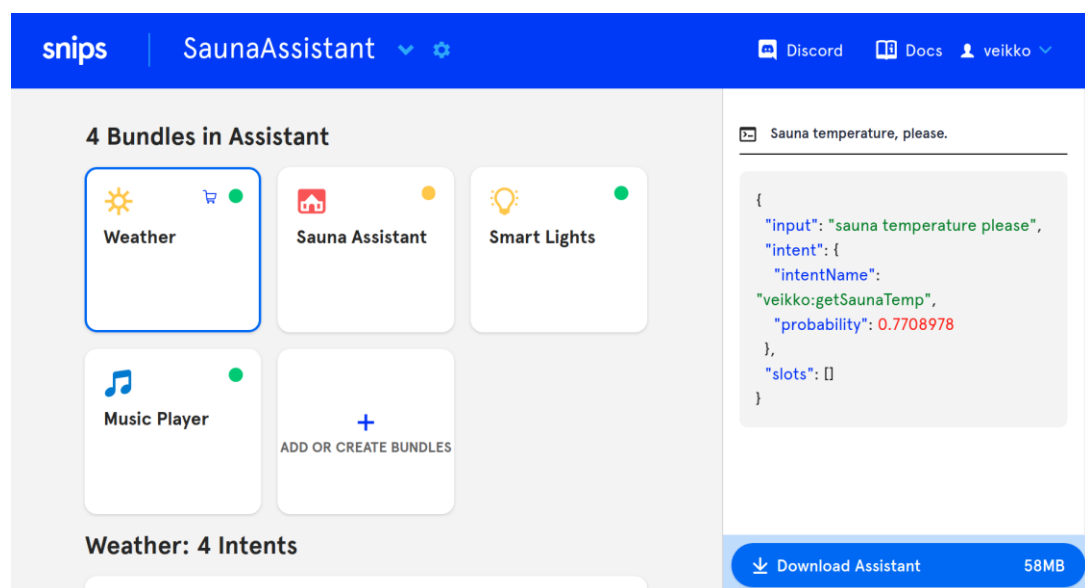


Figure 8. View from Snips Console

The Snips Voice Platform is the core of operations while inside the sauna, as the only way to interact is with voice, since use of a smartphone is hard and even dangerous, as the battery could easily explode under heavy stress of high temperatures. The microphone inside the sauna can be activated and deactivated to disable unwanted hotword detection during normal conversation, for example. The intents captured by Snips are handled in the Home Assistant.

5.4 Mosquitto and MQTT

Eclipse Mosquitto is an open source, lightweight message broker that uses MQTT (Message Queuing Telemetry Transport) protocol that can be used on low power devices or in large servers.

The MQTT is used to publish and subscribe messages to different topics. The configuration for said topic is not needed, as the topic is created on the first publish message. The clients connect to the same broker and use the subscribed topics as seen fit. Sending data to database over the MQTT allows for quick additions of new data points to publish on same topic (Light.)

The MQTT sends data from the sensors read by the Node-RED directly to Home Assistant. Home Assistant shows the latest data gathered and this data can be used to automate functions or by Snips to enquire, for example, sauna's temperature by voice. One broker handles all the traffic coming to the MQTT, Snips AI uses the same way to communicate between its different services. The broker can be protected with user credentials, in which point Snips fails to operate, for the credentials cannot be implemented to be used by Snips, at least at the time of writing. Because of this, the broker is usable and accessible only on the localhost for a low level of protection.

5.5 Mopidy

Mopidy is a music server written in Python, it can be used straight from the terminal or as a background service on Linux and Mac based devices. Normally functioning as MPD (Music Player Daemon) and a HTTP server, Mopidy can also

be implemented with additional frontends (Figure 9) to better suit needs. Mobile applications can also be used for better access to the MPD.

As vanilla version, Mopidy can be used to play music from the local disk and from the radio streams. With extensions, Mopidy can play music from different online music sources, such as Spotify, or to stream straight from video sources, for example, YouTube. The most popular device to run Mopidy on is the Raspberry Pi, though on older versions the service can occasionally be rather slow (Mopidy 2018.)

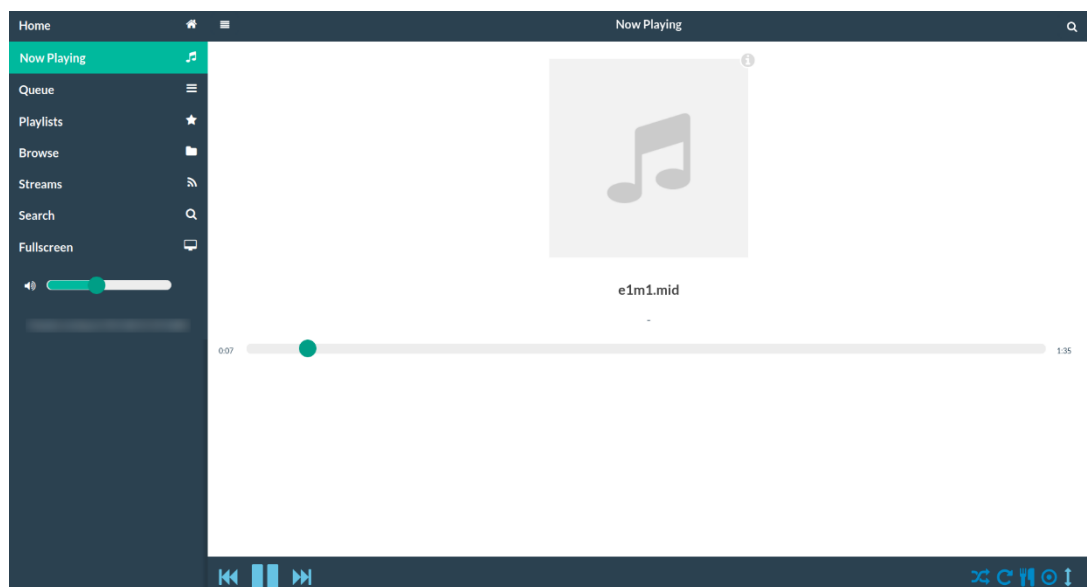


Figure 9. Mopidy Musicbox Webclient user interface

Mopidy is used to play music from different sources, with the default source being the Spotify. Spotify searches can be made within the Mopidy for artists, albums and playlists, with user's own created playlists available as well. Snips AI can be used to control the Mopidy service and the music play, such as changing to the next or previous songs and controlling the volume. Selecting the playlist via voice input is not possible at the time; however, as the Snips intents refuse to communicate with Mopidy as such. The playlist may be selected from a mobile application used with e.g. a smartphone located on the outside wall of the sauna structure, as bather enters the sauna.

5.6 Festival TTS

Text-To-Speech (TTS) is a technology where the written, normal language text is transferred into synthesised speech. There are numerous TTS services available for many different languages but for Finnish not so many. The best candidate after a quick research was the Festival Speech Synthesis System and its Finnish language library *Suopuhe*.

The Festival was developed at Centre for Speech Technology Research (CSTR) in the University of Edinburgh to address the need for better speech synthesis. The Festvox is a platform for the Festival to build new voices and languages on. The Festival and Festvox have preinstalled languages, with the most advanced one being the English language. By default, good coverage for French and Spanish. Building a new voice with the Festbox for English language will take a few days. However, building a new language and a voice for it can take months or even years to achieve a proper level (Festival – Festvox 2017.)

Suopuhe was a project funded by TEKES and was developed for speech synthesis of Finnish language in 2003. *Suopuhe* consists of two voices, male and female, and the voice generating is done with pre-recorded phone pairs that are modelled with linear prediction used in mobile phones, a method called diphone synthesis (Loponen 2005.)

5.7 DigitalOcean

DigitalOcean is used as the main cloud infrastructure platform, or server, for the setup. DigitalOcean is an IaaS (Infrastructure as a Service) provider, hosting virtual machines for the service buyers to use. DigitalOcean offers machines of different sizes and power for different purposes, such as running a database. Services can be scalable, scaling according to the current need of said service. DigitalOcean comes with several pre-built distributions and applications, which can quickly be fired up in, e.g. testing purposes (DigitalOcean 2018.)

The cloud services installed and operating in DigitalOcean are the InfluxDB database and visualisation and monitoring tool Grafana. Both services are described more specifically below, with installation instruction in the section of building the solution.

The DigitalOcean machine runs on Linux Ubuntu 16.04, providing 1GB of RAM and 25GB of storage with 2.4GHz processor, which should be more than enough for the cloud, since the database and Grafana are not huge on the memory side. The data incoming to the database is small and the storage for this purpose should last for a long time.

5.7.1 InfluxDB

InfluxDB is a light-weight, open-source time series database written in Go programming language. Designed for fast and high-availability storage for timestamped data, the InfluxDB is useful when used to store IoT device data, which usually comes in large amounts. With the DigitalOceans machine the InfluxDB can do approximately 5 000 field writes per second. InfluxDB uses queries similar to SQL (Structured Query Language) (InfluxDB 2018.)

The InfuxDB can be accessed through the command-line interface (CLI) to manage the database as either normal user or admin. The before supported Admin UI, accessible from the browser, is no longer available after the InfluxDB v1.5 and the changes to the InfluxDB service or the database must be done on the CLI side as an admin user (Figure 10).

```
Connected to https://localhost:8086 version 1.5.1
InfluxDB shell version: 1.5.1
> auth
username: someusername
password:
> USE dataBase;
>
```

Figure 10. InfluxDB command line interface

The measurements, or tables, are created as the data first arrives to the database from the Node-RED nodes. Each node represents own measurement for the specific data sent.

5.7.2 Grafana

The data is sent to be monitored and visualised in Grafana, an analysis and visualisation tool designed for timeseries databases to make the incoming data more readable by the user.

Grafana offers different in-built visualisation options, varying from normal timeseries graphs to single stat gauges (Figure 11). The user can define alerting rules for important data, such as in this case for temperature readings, should the temperature rise above the set level a message can be sent to the user to notify about the status inside the sauna. The dashboards created inside the Grafana can be made more dynamic with the help of templates and variables, changing the values to show data from a different source in the same dashboard for example (Grafana Labs 2018.)

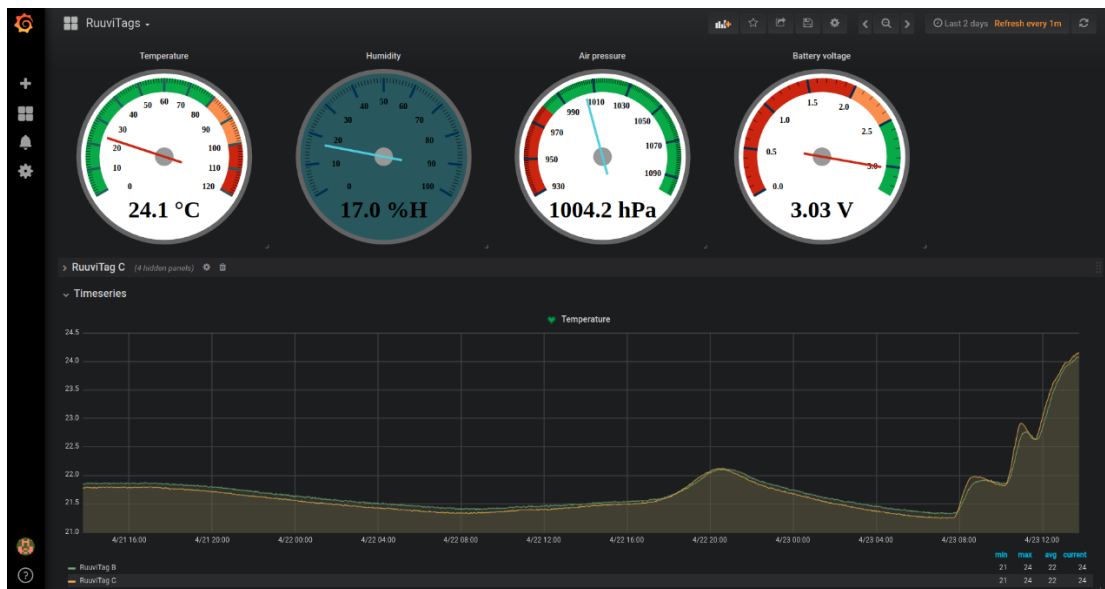


Figure 11. Dashboard view of RuuviTags in Grafana

Grafana shows the data from InfluxDB timeseries database. Multiple dashboards are created to show data from different sensors and/or measurements. Some values are also viewable through the same dashboard with the use of variables, though the whole is easier to see when sensors are shown separately on own tables and gauges.

6 Hardware

The hardware was carefully selected from vast variety of possibilities. The hardware gathered for the thesis work had specific requirements; to be able to function without any errors under high temperature and humidity loads, as well as to withstand the potential harsh environment surrounding the sauna itself.

The following chapters describe the hardware selected, including the sensors and other physical devices implemented and connected to the sauna. Some of the devices are predefined, given to the writer's use by the client from personal collection, while other devices and components were either bought locally or ordered through the Internet from global suppliers.

6.1 Raspberry Pi

Raspberry Pi (Figure 12) is a small and affordable single-board computer designed for different user projects. These devices can be used as any other computer, provided all the power that a bigger computer delivers is not needed. First introduced in 2012 with the initial version, the Raspberry Pi has sold over 12 million devices, with all the version included, in its five-year career, according to Circuit Breaker blog in March 2017. (Miller 2017)



Figure 12. Raspberry Pi 3 Model B used in the solution

Raspberry Pi by default runs on Debian based Raspbian operating system. The version used in thesis, the Raspberry Pi 3 Model B (Figure 12), is the most sold version at the time of writing. It delivers enough power to handle all the needed components for the project. With built in Bluetooth and WiFi connections it can also operate on the Internet and connect to a speaker without the necessity of additional cables. Technical specifications of the Raspberry Pi 3 Model B can be seen in the figure (Figure 13).

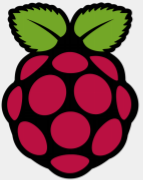
	Raspberry Pi 3 Model B	Raspberry Pi Zero	Raspberry Pi 2 Model B	Raspberry Pi Model B+
Introduction Date	2/29/2016	11/25/2015	2/2/2015	7/14/2014
SoC	BCM2837	BCM2835	BCM2836	BCM2835
CPU	Quad Cortex A53 @ 1.2GHz	ARM11 @ 1GHz	Quad Cortex A7 @ 900MHz	ARM11 @ 700MHz
Instruction set	ARMv8-A	ARMv6	ARMv7-A	ARMv6
GPU	400MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV
RAM	1GB SDRAM	512 MB SDRAM	1GB SDRAM	512MB SDRAM
Storage	micro-SD	micro-SD	micro-SD	micro-SD
Ethernet	10/100	none	10/100	10/100
Wireless	802.11n / Bluetooth 4.0	none	none	none
Video Output	HDMI / Composite	HDMI / Composite	HDMI / Composite	HDMI / Composite
Audio Output	HDMI / Headphone	HDMI	HDMI / Headphone	HDMI / Headphone
GPIO	40	40	40	40
Price	\$35	\$5	\$35	\$35

Figure 13. Raspberry Pi specifications for different models (Benchhoff 2016)

In whole the Raspberry Pi functions as the core for processes and services in the final solution, being responsible of gathering the data from numerous sensors and then sending it through the Internet into a cloud service to be stored and monitored as one sees fit.

6.2 Sensors and components

The used components and/or devices are listed in the table (Table 2) and are later referred to with the ID of said entry. The table is also shown in Appendix 4 for easier access.

Table 2. Used components and parts

ID	Function	Description	Location
----	----------	-------------	----------

M1	Moisture	DHT22	Upper left wall
M2	Moisture	DHT22	Lower left wall
T1	Temperature	DS18B20	Left wall
T2	Temperature	DS18B20	Concrete, below stove
T3	Temperature	Thermocouple K-type	Roof, above stove
T4	Env. sensor	RuuviTag	Air income pipe
T5	Env. sensor	RuuviTag	Roof insulation
SP1	Speaker	UE Roll 2	Under sauna benches
MIC1	Microphone	USB microphone	Close to bather
US1	Ultrasonic	AJ-SR04M	Top of water tank
RL1	Relay	8-channel board	Fieldbox
CON1	Converter	Step-Down converter	Fieldbox
BT1	Button	Microphone activation	Close to bather
BT2	Button	Emergency kill switch	Wall
TB1	Term. block	Terminal block	Fieldbox
CB1	Circuit board	Circuit board	Fieldbox
FB1	Fieldbox	Fieldbox	Sauna back end
WP1	Water pump	Water pump	Near water tank
L1	Light	Light 1	Left wall
L2	Light	Light 2	Back wall

Sauna is fitted with some casings, which can be used as installation points for sensors inside the sauna, e.g. for the humidity sensors (Figure 14). For a visual view of the hardware placement see the Appendix 5.



Figure 14. Example of sensor casing

6.2.1 Temperature sensors

A large amount of the sensors are temperature sensors. These sensors vary in preferred measuring areas and are located accordingly inside the sauna. With the normal temperature inside the sauna being somewhere between 70 and 100 degrees Celsius, these sensors have been selected with a high temperature operating range in mind.

Most of the sensors can measure up to 125°C and after that should be functional to at least 150°C. However, continuous exposure to such high temperatures may influence the sensor. Because of this, all the sensors inside are made to be easily replaced, should one or more fail.

The pool of available sensors was defined by the preferred use of digital sensors, because of the Raspberry Pi's digital GPIO pins, allowing measurements without AD-converters. Another definition was that only three wires were possible to be used for the sensors, due to the cables running inside the sauna. Use of only

digital sensors limits the operating range of said sensors, analogue sensors being able to sense higher temperatures. The temperature of +125°C was still adequate for the sauna environment.

DS18B20 (T1, T2) sensors are relatively small and effective digital temperature sensors. These sensors operate between -55°C and +125°C, with the accuracy of $\pm 0.5^\circ\text{C}$ between -10°C and +85°C. Outside this range the accuracy may decrease to $\pm 2^\circ\text{C}$, which in the case of a sauna is adequate.

Sensors come in two variations, one version being open to the elements and other encased (Figure 15). The latter one is surrounded with stainless steel metal casing with PVC wrapped around the three wires of the sensor, ensuring the device remains waterproof. Because of the PVC plastics nature this version is prone to some deformation under certain temperatures, starting around 100°C, if applied too much pressure, and the final total melting point being close to 200°C. In used environment these temperatures should not occur under any normal circumstances, with the highest temperature being present just above the stove, where another sensor with higher heat resistance is used (PVC Heat Distortion Temperature.)



Figure 15. Both versions of the DS18B20 sensor

The DS18B20 sensors are used to measure the basic temperature of the sauna, much like a normal analogue temperature gauge would do. Sensor has three wires, GND, data and VCC, with either 3.3V or 5V input. DS18B20 can also operate without external power input, in so called 'parasite-mode', where the power is drawn directly from the data pin, thus operating with only two pins. While this wiring method may sound intriguing, it is advised to use external power source, as it will produce more precise measurements, for the parasite-mode may cause excessive warmth issues with the sensor. As it is possible to use three pins for the sensor in this case, the normal connection is used.

A resistor of 4.7k to 10k is needed between the external power and data pins, same resistor can be used for multiple DS18B20 sensors. The DS18B20 also has an advantage over many other sensors; it has a 64-bit serial code, which means that multiple sensors can be connected via the same data pin and identified by their unique code. This can be convenient in larger solutions, where the need for numerous sensors is present.

Thermocouple (T3) sensors are temperature sensors, which can operate in relatively high temperatures. The sensor consists of two electrical conductors of a different nature, forming electrical junctions based on the current temperature. The voltage varies accordingly, thus giving the possibility of high accuracy temperature readings.

The thermocouple sensor used in the setup is of a K-model (Figure 16), which is capable of measuring temperatures approximately between 0 and 1024 degrees Celsius, with sensing accuracy of 0.25°C. This model is formed with chromel-alumel junctions, giving it a high temperature range for the usage in the sauna environment. The normal temperature of a sauna varies between 70 and 100 degrees Celsius. In this case, however, the sensor is installed in the ceiling above the sauna stove, where the temperature can exceed the preferred sensing area of other used sensors, which could result in a lowered life expectancy or even a breakdown of said sensors. The Thermocouple sensors use a special thermocouple wire, extension cables being available for purchase if necessary. In this case, the sensor is connected via a normal cable to the Raspberry Pi, as the

thermocouple wire is not long enough. This can influence the accuracy of the sensor if used on longer distances, but under the distances in a sauna the accuracy should remain approvable.

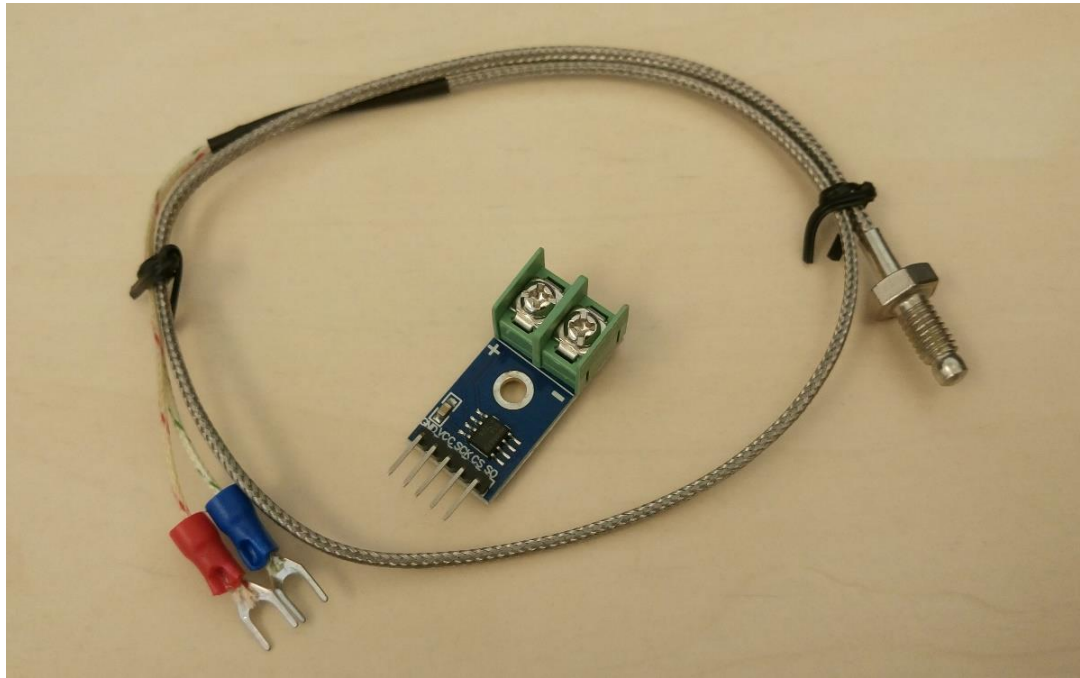


Figure 16. Thermocouple K-type with MAX6675 converter

Thermocouple sensor needs a Thermocouple-to-Digital converter, which in this case is provided by MAX6675 converter. The converter can be directly connected into Raspberry Pi 3 via its five pins, with VCC of 3.3V and GND coming from the circuit board. The converter does need a driver to operate, which can be found as open-source from Github, licensed free to use and modify as seen fit, provided the said license is included in made variations as well. (Hefnawi 2014)

The driver is easy to use, with an available example code to run. The example code reads the measurement from the thermocouple, from specific GPIO pins, and prints it on the terminal. One addition needed to be made into original code, however, as it would not function correctly without it. Before reading the sensor reading the software needs to sleep for at least 200 milliseconds, as stated in the issues of said Github page. With no sleep time between the initialisation and reading, the sensor could send either zero or last successfully measured reading, thus making the sensor useless.

6.2.2 Humidity sensors

In addition to temperature sensors, the other important sensor type is the humidity sensor. Humidity sensors traditionally measure the relative humidity readings from the surroundings. Inside the sauna, this reading is important for the well-being of the bather and sauna as well.

A normal absolute humidity, depending on the temperature in the sauna, is somewhere between 30 and 40 grams per a kilogram of air, which as relative humidity percentage is between 10% and 30%. To achieve the specific desired humidity in sauna, water must be applied accordingly on the stove. Higher temperature requires higher amount of water applied (Lämpötila ja löyly 1997.)

DHT22 (M1, M2) is the primarily used relative humidity sensor, which can also measure temperature readings (Figure 17). DHT22 has four pins, with 3.3V to 5V power input, data pin and GND, one of the pins being a no-connect one. Sensor comes with a normal 4.7k to 10k-pullup resistor, which connects between VCC and data pin, much like described before with the DS18B20 temperature sensor.

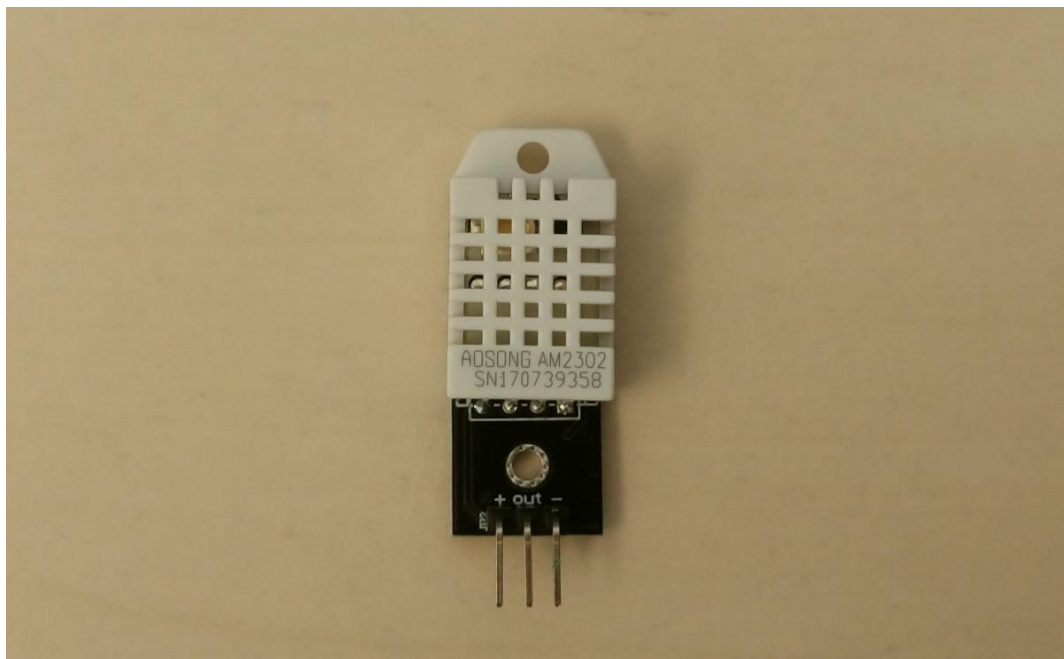


Figure 17. DHT22 temperature and humidity sensor

In the solution the sensor has an additional board connected into it, with the required resistor built in, thus leaving only three pins to be connected directly

into Raspberry Pi 3. These sensors are in the inside wall panelling off the sauna on different heights to get accurate readings on different levels.

6.2.3 RuuviTag

RuuviTag (T4, T5) is an open-source sensor beacon platform that can measure temperature, humidity, air pressure and acceleration in X, Y and Z axes (Figure 18). By default, the RuuviTag can be used as a small weather station with Eddystone or iBeacon features but it can also be used to more complex solutions. The RuuviTag uses BLE (Bluetooth low energy) technology to provide the data for devices within the broadcasting range.

RuuviTag is usable straight from the box. It comes with preinstalled firmware and a battery of 3V (CR2477), which is said to last somewhere between 5 to 10 years, depending on the environment and RuuviTag's mode. Should the battery fail sooner, the holder allows for quick battery replacement. The software is easy to update, no physical connection between the tag and a computer is needed as it can be flashed from a smartphone. In basic URL mode it can measure temperature, relative humidity and air pressure, acting as, for example, an Eddystone proximity beacon, with the data accessible in the Eddystone URL (RuuviTag.)



Figure 18. RuuviTag with waterproof casing

In thesis setup the RuuviTag is set to RAW mode, which in addition to measurements of above, can also collect acceleration and battery voltage data. RuuviTag can measure temperatures roughly between -40°C to $+85^{\circ}\text{C}$ but is recommended to be maintained from -20°C to $+60^{\circ}\text{C}$ for longer life expectancy and battery life. The casing around the RuuviTag is waterproof and the whole has been tested multiple time in saunas and even in boiling water at around 100°C with positive results (Ruuvi: Weather Station 2016/2017) (Official RuuviTag Firmware 2017.)

In sauna the RuuviTag is not solely used to measure data from inside the sauna as other sensors are but instead to measure the incoming air temperature and with the possibility to measure temperatures inside the ceiling insulations as well to monitor that the temperature there stays within accepted range. Both locations are easily accessed for the device to be removed or replaced.

6.2.4 Ultrasonic sensor

The ultrasonic sensors (US1) send out signals to ping an object and measure the time between the signal sent and signal received, thus creating the measurement of distance.

In thesis work, the AJ-SR04M waterproof ultrasonic sensor (Figure 19) is located on the top half of the water tank under the sauna. With the measurement from the ultrasonic sensor the amount of water can be measured and used to determine, whether it is safe to automatically, or manually, throw water on the stones on the stove. The measurement range of the sensor is between ~ 20 to ~ 400 centimetres.



Figure 19. Waterproof ultrasonic sensor AJ-SR04M

The sensor is read with C++ code and ran at specific intervals with other sensors in Node-RED service. The measurement is in centimetres, increasing as the water amount in the tank diminish. With the knowledge of the radius and height of the water tank, this measurement can be converted to litres for easier readings.

The measurements are accurate enough for the purposes of the thesis work. While the sensor reading may vary drastically on some occasions, the average value of those readings can be considered accurate. Analogue water gauge can be added, should errors start to happen with the ultrasonic sensor.

6.3 Sound

6.3.1 Speaker

As predefined piece of hardware the selected speaker (SP1) was UE Roll 2 (Figure 20). A small, yet big on sound speaker was a good choice for audio output. The speaker can be moved around the sauna, at least on the benches, as the connection between Raspberry Pi 3 and the speaker is handled with Bluetooth. The exact procedure to enable this is described in own section. While the Bluetooth connection is preferred standard, connecting the speaker via 3.5mm jack is also possible, provided the cable is long enough.



Figure 20. UE Roll 2 Bluetooth speaker

Speaker is IP67 rated, able to withstand humidity values inside the sauna, as well as total submerging in water. The high temperatures cause the outer casing of the speaker becoming hot but should not affect the speaker itself. The battery inside the speaker also depletes more quickly in high temperatures, as the one in a smartphone would, but because of the easy movability of the speaker, it can be removed from sauna when not needed, thus improving life expectancy.

6.3.2 Microphone

Finding a functioning microphone (MIC1) under the environment of a sauna is not easy. The microphones proofed to work under high temperature and protected against humidity are rare, at least from the cheaper category.

The best solution was to have a cheap and easily replaceable USB microphone (Figure 21), which can be connected either directly to the Raspberry or, in this case, via extension cable, so that the microphone could be located in a way that the speech can be heard clearly. The heat and humidity matter may influence the USB microphone; higher temperature and humidity may cause the microphone to stop working. The heat and humidity may also affect the USB ports directly, again resulting in a failure. Thus, both components, the extension cable and microphone are easily replaced if needed. The microphone used is made of ABS plastic, which should withstand higher temperatures than PVC (PVC Heat

Distortion Temperature) but as the microphone is not officially protected against the environment the life expectancy is hard to define.



Figure 21. USB microphone used in solution

The exact USB microphone model is hard to define, however, when using the *usb-devices* command, the manufacturer is stated being C-Media Electronics Inc. but is not confirmed knowledge. Any USB microphone should be compatible with the Raspberry Pi 3 and thus function correctly.

6.4 Fieldbox

Fieldbox (Figure 22) is located in the back end of sauna structure. The fieldbox is secluded from the temperature and humidity issues of the sauna interior and is where the more delicate devices are located. These devices are the Raspberry Pi 3, as described earlier, a relay board for using the lights and water pump, and a DC/DC step-down converter to be used for the Raspberry Pi.

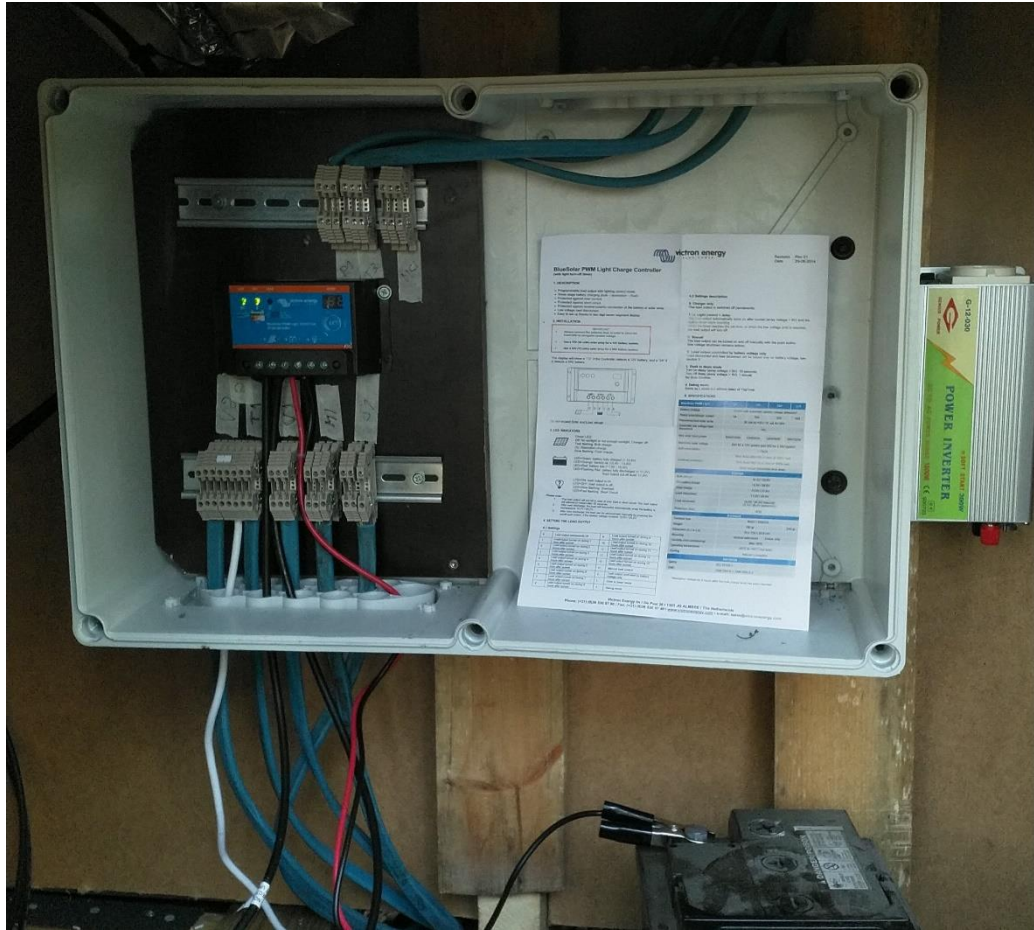


Figure 22. Fieldbox before installations with battery connected

The devices located in the fieldbox are attached on a panel, which in whole can be removed from the fieldbox for maintenance. This panel includes the Raspberry Pi, the circuit board (CB1), the relay board (RL1), the DC/DC converter (CON1) and the terminal block for the cables (TB1).

A **relay board** (RL1) is used to handle the additional appliances, such as lights and water pump. The specific relay board consists of 8 relays and accepts voltages from 5VDC to 250VAC (Figure 2). A car battery (12V) is located under the sauna, which provides the needed power for the appliances connected to the relays to function.

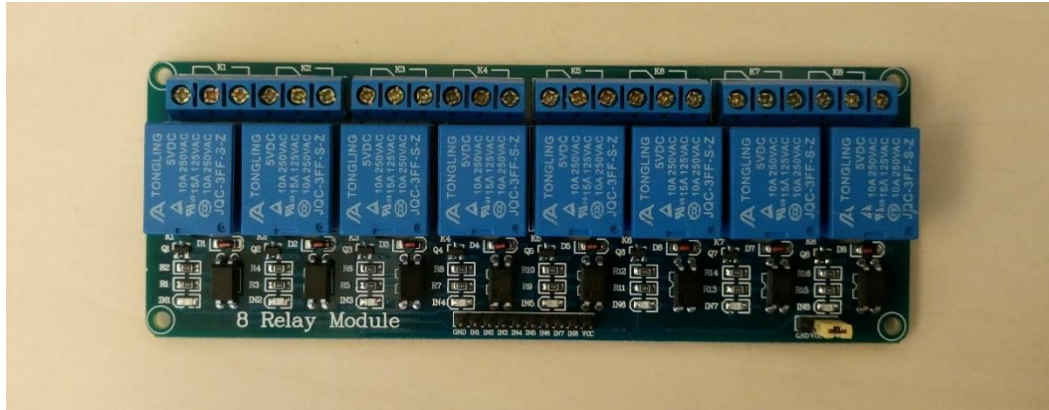


Figure 23. 8 Relay Module used in solution

The relays are only used from the Home Assistant directly, configured in the Home Assistant's configuration file with corresponding GPIO numbers for each relay to initialise. No external driver is needed to operate the relay board, only specific GPIO numbers in the Home Assistant or Node-RED.

DC/DC step-down converter (CON1) (Figure 24) is used to convert the power supply to accepted level for the Raspberry Pi 3 from the 12 volts of a car battery. The Raspberry Pi normally uses the official power supply of 5V/1A but with the car battery the power input would be enough to burn the circuits and render the Raspberry useless. The DC/DC converter lower the voltage from 12 V to 5V and the amperage to 3A maximum. As with other electric devices the voltage is the more important part with the power supply as the Raspberry Pi only takes as much amps as needed.



Figure 24. DC/DC converter used in solution

With the used converter the input voltage of 12 volts is connected to the two wires seen in the figure. The output of 5V is delivered through the female USB port to the Raspberry Pi 3. An extension USB cable is needed to connect between the converter and the Raspberry Pi's microUSB port.

The cables running inside the sauna connect to the **terminal block** (TB1), from where the connection is handled with jump wires to the Raspberry Pi 3. The relay board is connected to the Raspberry Pi with jump wires, the cables run from the relays to the lights and water pump, with 12V power supply applied.

Circuit board (CB1) (Figure 25) built for the solution provides the power supply for the sensors, both the 3.3V and 5V, for the relay board, of 5V, and for the two switches, or buttons, both of 3.3V. The DS18B20 temperature sensors (T1, T2) connect to a 4.7k resistor between the power pin and data pin. Same resistor can be used for several DS18B20 sensors. The buttons, one to set microphone either on or off and other to act as a kill switch in case of emergency, are both behind a single 4.7k-pulldown resistor.

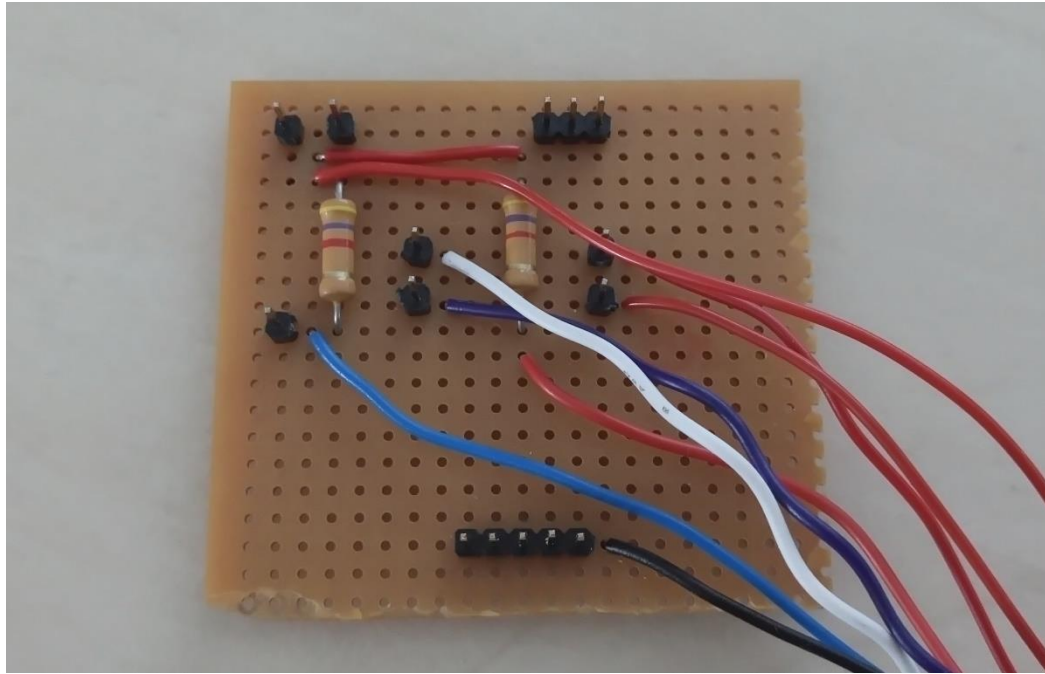


Figure 25. Circuit board constructed for the setup

7 Building the solution

This chapter describes the process of building the solution and platform for the project in its entirety to offer an understanding of what was achieved and executed during the thesis work. The building of the actual system, with the components and devices described before, as well as the installation and configuration of the needed tools and services, are explained in the following chapters.

7.1 Software architecture

The software architecture is explained for a better understanding of why the certain tools were selected amongst many. The flowchart (Figure 26) simplifies the architecture, showing the connections between certain larger parts of the system, with smaller, yet as important parts being explained.

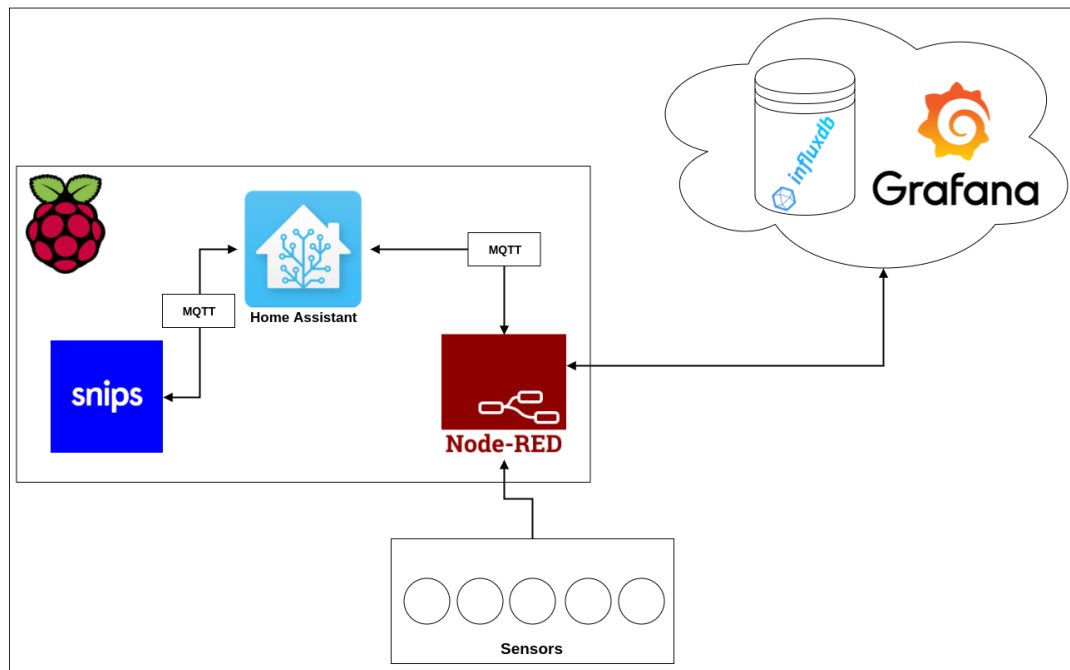


Figure 26. Flowchart of the system

The core for the system is provided by the Raspberry Pi 3 Model B. With the requirement (RE100) of being able to function independently with or without the Internet connection, at least limitedly on the latter part, the used tools were selected accordingly. One tool addressing this requirement was the Node-RED.

The idea of using the Node-RED as the controller for the sensors and components connected to the Raspberry Pi, as well as for the connection between the device and the server, occurred when first conversations of the project were held with the client. Node-RED could easily be used to handle the codes and scripts for different purposes on the device, with a great selection of user-made nodes for different components, such as the used sensors. This meant that no external and additional code had to be constructed for the sensors to work. The same applied also to the database connection, what normally would be done with either C++ or Python was now handled by Node-RED. For some components additional scripts or codes had to be implemented, one being the button for applying water onto stove (RE200). The automated Internet connection was controlled with the Node-RED, as Raspberry Pi may drop the WiFi connection at certain points or the sauna could be moved to a different location, where the previous network is not available.

Another large requirement for the system was to be able to control some of the components with voice commands (RE210, RE300, RE301, RE302, RE410, RE510, RE511, RE512). Many speech recognition services are available, e.g. the Google Assistant's API.ai or Amazon Echo's Alexa. Both of the named need an access to the cloud services, which requires continuous Internet connection. Should these services be used in the sauna, voice control system would fail immediately in case of a network error. Therefore, a service that could function without the Internet connection was needed. The Snips AI was chosen for largely growing popularity and for its private-by-design and on-device principles, as well as for a good documentation regarding the installation and usage. The Snips AI communicates internally between its different services with MQTT messages, which are also used to communicate with other services running on the local device.

The Home Assistant was first noticed during the conversations with the client at the start of the project. Home Assistant was recognised as a great choice for the automation and remote controlling of the system, due to its simple UI, which allowed many possibilities for remote controlling (RE400), and easy-to-make automations. Creating configurations for the Home Assistant proofed easy and addressed many of the requirements together with the Snips AI, being able to handle all current and to-be automations (RE201). The Home Assistant is used to access the current data, which is sent to MQTT broker from Node-RED. The Snips AI use this data from the Home Assistant to provide information to the user when desired. The Mopidy Music Player is implemented into Home Assistant to access and use the service from either Home Assistant's UI or through Snips AI (RE500).

The cloud infrastructure is provided by DigitalOcean, as it turned out to be the best choice for IaaS service for the project. The database, InfluxDB, was chosen as the database due to its light weight and compability with timeseries data from IoT devices. The Grafana was selected as the visualisation tool for the sauna data, mainly for the easy usage and monitoring. Grafana was seen as sensitive choise for data analysation and visualisation, as former knowledge of the software was present.

The following chapters describe in detail the installations, configurations and usage of tools and hardware alike, in order to achieve the setup for the system.

7.2 Raspberry Pi

7.2.1 Installation and backup

The installation of the Raspberry Pi 3 is an easy enough procedure to do as there is a fine documentation found on Raspberry's own homepage. The mainstream installation, however, was as follows.

A fresh, out of the box Raspberry Pi 3 usually comes with the micro-SD card already including the installer software. This version is referred as NOOBS (New Out Of the Box Software) and is the easiest way of installing. This method is not available for the installation of the image used in thesis. The image used is Hassbian, and as opposed to the Raspbian image, comes with no graphical user interface (GUI) possibility.

The Hassbian Image is found on the Home Assistant's own homepage and needs to be flashed to the micro-SD card. This can be done with the use of Etcher, a very user-friendly flashing software. Some machines have no option to insert the SD card for flashing directly. In such cases only way to do it is to use a USB flasher and connect the SD card via the USB port (Install Home Assistant.)

Flashing will take from approximately 5 to 10 minutes. Once done, the micro-SD card can be connected into the Raspberry Pi 3, with correct Raspberry Pi power supply. While starting the Raspberry Pi for the first time it is advised to have a monitor connected to the Raspberry Pi via the HDMI output. The monitor can be used throughout the system building if possible but SSH connection can also be used for the same purpose from another computer.

At some point, it is wise to copy the micro-SD card of the Raspberry Pi 3 in case of data corruption or physical breaking of either Raspberry Pi or the micro-SD card itself. The `dd` command can be used to make an image of the micro-SD card to another computer. As before, the card is inserted either in the internal card

socket, or if not available, to the external USB flasher and connecting it to the computer.

```
# See the name of the SD card filesystem
$ df -h

# Locate SD card, in this case /dev/sdb2
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb2       15G   5,8G   8,2G   42% /media/.../3f...

# Make image of the card
$ sudo dd if=/dev/sdb2 of=/home/user/rpi_image.img bs=1M
```

The command above makes an image of the specified SD card and saves it to the specified directory with given name. The *bs* argument is used to control the block size of read and write operations. The operation can easily take up to 15 minutes on 16GB micro-SD card. The length depends on the speed of the USB drive and the size of the micro-SD card.

7.2.2 Internet connection

The Raspberry Pi is connected to the Internet via the Ethernet cable or with WiFi connection, the former is easier as it requires no additional configuration. The wireless connection is configured in */etc/wpa_supplicant/wpa_supplicant.conf*.

```
# List available networks
$ sudo iwlist wlan0 scan
# Edit /etc/wpa_supplicant/wpa_supplicant.conf
...
network={
    ssid="ESSID_OF_NETWORK_1"
    psk="PASSWORD_FOR_NETWORK_1"
    priority=1
    id_str="network_1"
}
network={
    ssid="ESSID_OF_NETWORK_2"
    psk="PASSWORD_FOR_NETWORK_2"
    priority=2
    id_str="network_2"
}
...
```

The *wpa_passphrase* can be used to generate a more secure password and to remove the plain text password.

```
$ wpa_passphrase "network_ssid" "verySecurePassword"
```

This prints the configuration on the screen, from where the *psk* can be copied to the *wpa_supplicant.conf* file.

The file above is used in the */etc/network/interfaces* to have either static or DHCP (Dynamic Host Configuration Protocol) distributed IP address. By default, the IP is distributed by the DHCP and the configuration file is as follows:

```
# Edit file /etc/network/interfaces
...
auto lo
iface lo inet loopback

auto eth0
allow-hotplug eth0
iface eth0 inet dhcp

auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
...
```

The sauna structure is fitted with a router to be used when the home network is not reachable. The priority of connection is for the home network and secondly for the router inside the sauna itself. The prioritisation is done in the */etc/wpa_supplicant/wpa_supplicant.conf*. The router configuration is not handled in thesis as the network providers offer guides for various routers and a specific guide is not needed in this case.

To access Raspberry Pi from an external computer the SSH (Secure Shell) can be enabled from the *raspi-config*, as SSH is usually disabled by default. At this point, if not done already, the default password should also be changed into more secure one.

The Raspberry Pi has a hostname given through the installation of the Hassbian image, the default hostname being in this case *hassbian*. This can be changed to

something more suitable by consulting two files, */etc/hosts* and */etc/hostname*. This name can be used to access the services running on the Raspberry Pi through the browser by giving the port to the specific service, such as in this case *sauna.local:8123/* would be the address for the Home Assistant service inside the local network. This makes it easier to connect to the services without the knowledge of the current IP address of the Raspberry Pi.

During longer times of uptime, with the Raspberry Pi active but not connected to the router via WiFi connection, the WiFi on the router entered a stand-by mode, similar to a sleep mode. Wired connection worked with a computer from the router, however, connecting the Raspberry Pi to the router with WiFi failed. This was also tested with a mobile device. Restarting the router solved the problem. This problem should not occur normally, as the Raspberry Pi should always be connected to the router, if no home network is available in range, thus the router should not enter sleep mode.

The Raspberry Pi's WiFi module may at some point enter sleep mode, in which case the *wlan0* interface needs to be restarted. This can be done with following commands.

```
$ sudo ifdown wlan0
$ sudo ifup wlan0
# Alternatively
$ sudo service networking restart
```

Should the restart of the interfaces fail for some reason, e.g. due to modifications made to the */etc/wpa_supplicant/wpa_supplicant.conf*, the *networking* service can also be restarted, as this reloads the configuration files for network interfaces.

7.2.3 Configuration

Several configuration files are modified during the setup of the system. These files, with the services in question, are handled on that specific service's section of the following chapters.

The GPIO pins on the Raspberry Pi are not readable by default, as additional software needs to be installed for this purpose. To read and use the GPIO pins the *wiringPi* software is cloned and built.

```
# Clone wiringPi
$ git clone git://git.drogon.net/wiringPi
# Build wiringPi from the directory
$ ./build
```

After the steps above the GPIO pins can be read with the *gpio readall* command. This prints the name and status of each individual pin, as well as the physical, wPi and the BCM pin numbers, as seen in the figure (Figure 27).

Pi 3											
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
		3.3v			1	2		5v			
2	8	SDA.1	IN	1	3	4		5v			
3	9	SCL.1	IN	1	5	6		0v			
4	7	GPIO. 7	IN	1	7	8	0	IN	TxD	15	14
		0v			9	10	1	IN	RxD	16	15
17	0	GPIO. 0	IN	1	11	12	0	IN	GPIO. 1	1	18
27	2	GPIO. 2	IN	1	13	14		0v			
22	3	GPIO. 3	IN	0	15	16	0	IN	GPIO. 4	4	23
		3.3v			17	18	0	IN	GPIO. 5	5	24
10	12	MOSI	IN	0	19	20		0v			
9	13	MISO	IN	0	21	22	0	IN	GPIO. 6	6	25
11	14	SCLK	IN	0	23	24	0	IN	CE0	10	8
		0v			25	26	1	IN	CE1	11	7
0	30	SDA.0	IN	1	27	28	1	IN	SCL.0	31	1
5	21	GPIO.21	OUT	1	29	30		0v			
6	22	GPIO.22	OUT	1	31	32	0	OUT	GPIO.26	26	12
13	23	GPIO.23	OUT	1	33	34		0v			
19	24	GPIO.24	OUT	1	35	36	1	OUT	GPIO.27	27	16
26	25	GPIO.25	OUT	1	37	38	1	OUT	GPIO.28	28	20
		0v			39	40	1	OUT	GPIO.29	29	21
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	

Figure 27. Raspberry Pi 3 GPIO pins as seen from *gpio*

The *gpio* command is mainly used to check the pin numbers, as the nodes for the sensors in Node-RED may use different number of each pin, for example the DHT22 (M1, M2) node uses the BCM number while the Ultrasonic sensor's (US1) C-code uses the *wPi* number. The *gpio* shows different GPIO numbers than the Raspberry Pi 3 header figure (Figure 28), the BCM value is equivalent to the GPIO numbers in the latter figure.

The DS18B20 (T1, T2) temperature sensor support the 1-wire technology and the same *data* pin can be used to read several sensors. The default pin number for this purpose is PIN 7 and is configured in the */boot/config.txt* file in Raspberry Pi.

```
# Add to the end of file in /boot/config.txt
...
dtoverlay=w1-gpio
# To change pin number
# dtoverlay=w1-gpio,gpiopin=x
...
```

This addition allows the Raspberry Pi to use the temperature sensor's 1-wire functionality, as specified earlier on said sensors chapter (See Chapter 6.2.1).

7.3 Connecting the hardware

The hardware described earlier consists of different devices and appliances. Together with the tools used in the solution, the system is made to operate independently, monitoring the surroundings of the sauna and outside world and to be controlled by the user when needed and seen appropriate. The GPIO layout is referred to in the hardware installation (Figure 28).

Pin#	NAME		NAME	Pin#
01	3.3v DC Power	Red	DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)	Blue	DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)	Blue	Ground	06
07	GPIO04 (GPIO_GCLK)	Green	(TXD0) GPIO14	08
09	Ground	Black	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	Green	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Green	Ground	14
15	GPIO22 (GPIO_GEN3)	Green	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	Red	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Purple	Ground	20
21	GPIO09 (SPI_MISO)	Purple	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	Purple	(SPI_CE0_N) GPIO08	24
25	Ground	Black	(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)	Yellow	(I ² C ID EEPROM) ID_SC	28
29	GPIO05	Green	Ground	30
31	GPIO06	Green	GPIO12	32
33	GPIO13	Green	Ground	34
35	GPIO19	Green	GPIO16	36
37	GPIO26	Green	GPIO20	38
39	Ground	Black	GPIO21	40

29/02/2016

Figure 28. Raspberry Pi 3 GPIO layout (Raspberry Pi 3 Pinout 2016)

For a better view of the whole setup, the block diagram is found in the appendices (See Appendix 1).

7.3.1 Sensors

The sensors, as described before (See Chapter 6.2), are powered from the Raspberry Pi's power supply pins, either from the 3.3V pin or the 5V pin. All sensors, apart from the Ultrasonic sensor used for water level measurements, use the 3.3V power supply. The 3.3V GPIO pin is used for two DHT22 sensors (M1, M2), two DS18B20 sensors (T1, T2) and for the Thermocouple K-type sensor (T3). The Ultrasonic sensor (US1) uses the 5V power supply to get more accurate measurements.

Most of the sensors are located behind the terminal block (TB1), with the cables inside the sauna structure connecting to it, while the connection between the terminal block and the Raspberry Pi is handled with smaller wires.

The two humidity sensors, the DHT22, are connected to the 3.3V power supply delivered to the circuit board. These sensors have in-built resistors but can be read only from separate GPIOs. The sensors are connected to the GPIO 17 and 27 (See Appendix 1), physical pins 11 and 13. The DHT22 node for Node-RED uses the GPIO, or BCM, numbering for the sensors.

The primary temperature sensors, the DS18B20 (T1, T2), share the input power of 3.3V with the DHT22 humidity sensors. The 4.7 kOhm resistor is set between the power and data pins, both sensors using the same resistor. Unlike the humidity sensors, the DS18B20 sensors connect to the same data pin with 1-wire technology (See Chapter 7.2.3), which then connects to the Raspberry Pi's physical pin 7 (GPIO 4). No GPIO pin numbers are needed for Node-RED's node.

Thermocouple K-type (T3) has additional converter located inside the fieldbox. The cables running inside the sauna structure connect to the converter, connecting with the Thermocouple sensor's own cable at the other end. The power input of 3.3V is drawn from circuit board's pin, while the CS, SCK and SO connect directly to the Raspberry Pi's pins 15, 18 and 22 (GPIO 22, 24 and 25). The sensor is read with Python script executed in Node-RED, available on the project's Git repository. (Saunaxio – thermocouple 2018)

The Ultrasonic sensor (US1) is attached to the panel inside the fieldbox, with sensor cable running to the water tank, where the sensor is attached facing down to the water surface. The power of 5V is drawn from the circuit board, while the ECHO and TRIG connect to the Raspberry Pi's pins 16 and 32 (GPIO 23 and 12). Sensor uses C++ code to measure the distance, code is available in full in project's Git repository. (Saunaxio – ultrasonic 2018)

7.3.2 Other hardware

In addition to the sensors the Raspberry Pi, the circuit board, relay board, speaker, microphone, the switches and the DC-to-DC converter placement is also described. The layout can be seen in the fieldbox layout (See Appendix 2).

The hardware inside the fieldbox is attached to a panel, which is removable in case of maintenance. The Raspberry Pi is located centrally in a way the other surrounding hardware is easily connected to it with jump wires. The power supply of 3.3V and 5V is connected to the circuit board (CB1) with said wires to provide the input power for components connected to it (See Appendix 2).

The DC-to-DC converter (CON1) is attached to the panel, close to the Raspberry Pi's power input port. The converter's power of 5V/max. 3A is delivered via USB cable to the Raspberry Pi. The power cables to the converter are connected to a terminal block, from where the cables continue to the car battery of 12V.

From the terminal block the 12V power output continues to the relays on the relay board (RL1) to supply the appliances, the lights and water pump, with the needed power. The relay board is directly connected to the Raspberry Pi with all GPIO pins, with the power input of 5V from the circuit board. The relays from the first to last relay connect to pins 36, 38, 40, 37, 35, 33, 31 and 29 (GPIO 16, 20, 21, 26, 19, 13, 6 and 5). First three relays are used to control both of the lights and the water pump, the rest relays act as a reserve. Relays' initialisation, as well as the main usage, is done inside the Home Assistant, with toggling made possible in the Node-RED with the *gpio* command.

The circuit board (CB1) is attached close to other hardware inside the fieldbox, similarly to the Raspberry Pi, making use of jump wires easier considering the length of said wires. Circuit board serves as a connection point between the Raspberry and other components, including the terminal block.

The speaker (SP1) and the microphone (MIC1) are located inside the sauna, both being easily removed when so desired. The speaker is connected with Bluetooth connection, so no physical connection to the Raspberry Pi is needed.

Alternatively, the speaker can also be used via cable to the 3.5mm jack (See Chapter 7.8.2). Speaker can be placed as seen fit, with some distance to the microphone to minimize unintentional hotword detection. The microphone is connected to the Raspberry's USB port via an extension cable and attached close to user at own specified place.

For visual description of the setup, see Appendix 1 and 2.

7.4 Node-RED

The Node-RED is pre-installed in Raspbian, or Hassbian, images but is of an older version, which causes problems during the installation of some nodes and is advised to be upgraded. Upgrading the Node-RED, Node.js and npm is the solution to avert the problems from occurring.

7.4.1 Upgrading Node-RED

During the building of the system the Node-RED was upgraded to a newer version. By default, the Node-RED is installed in the Raspbian image, in this case Hassbian image, and is usable from the start. However, the older version distributed within the images had problems with some of the installed nodes, e.g. with the DHT22 sensor node.

The easiest way to remove all older version from the Raspberry Pi was to uninstall the Node-RED with all dependencies. This way no old version would remain to conflict with newer version.

```
$ sudo apt-get uninstall -purge nodered npm node
```

The files created by the Node-RED, the `~/.node-red/` and `~/.node-gyp/`, as well as `.npm/` were removed to make way for a clean installation. These folders are usually located in the user's home directory.

The newest stable version of Node-RED can be installed with the *apt* package manager. The npm is also installed in the same line. Node.js is installed inside the

Node-RED package. The *node* is installed with *npm* and upgraded to latest stable release, as some *npm* installation may require newer version.

```
$ sudo apt-get install nodered npm
# Upgrading latest stable release of npm globally
$ sudo npm install npm@latest -g
# Latest stable node
$ sudo npm install n -g
$ sudo n stable
# Check version of nodejs, node and npm
$ nodejs -v && node -v && npm -v
```

After the installation the Node-RED, *npm* and *nodejs* should all be in the latest stable version and thus work as intended for the rest Node-RED nodes to be added. Version number for Node-RED at the time of writing was v0.18.4, for *nodejs* v4.8.2 and for *npm* v5.6.0. The *node* version was v9.8.0.

7.4.2 Installing required nodes

Node-RED is used to gather the data from the sensors and to send it to the cloud database, to handle data transfer directly to Home Assistant for quick usage with Snips Voice Platform and also to show current data for the user, viewed from the mobile application or computer's web browser.

Adding nodes in Node-RED is made relatively easy with in-built *Manage palette*. Most of the nodes were installed using this method, as it is the easiest and also the safest way, as the Node-RED service and npm handles the installation and no potential errors during manual installation could happen. Not all nodes are available from the *Manage palette*, however, as for example the node to read data coming from the RuuviTags can only be installed manually and directly to the Node-RED's directory, since no 'official' is available at the time of writing.

The available nodes installed in addition to the base nodes from the Node-RED's palette directly are listed in the table (Table 3), with a short description of contents and the used version number. The nodes installed with this method are

installed and located in the `~/.node-red/node_modules`.

Table 3. Additional Node-RED node installations

<i>Node</i>	<i>Description</i>	<i>v.</i>
<i>dht-sensor</i>	DHT22 node, temperature and humidity	1.0.1
<i>ds18b20-sensor</i>	DS18B20 node, temperature	1.3.5
<i>hostip</i>	IP address	0.0.3
<i>influxdb</i>	InfluxDB connection and queries	0.2.1
<i>noble</i>	BLE scan for RuuviTags	0.3.0
<i>dashboard</i>	Dashboard/UI nodes for Node-RED	2.8.2
<i>openweathermap</i>	OpenWeatherMap node	0.2.1
<i>smooth</i>	Number rounding	0.1.0
<i>stoptimer</i>	Stoptimer to auto redeploy flows	0.0.7

The *influxdb*, *noble* and *stoptimer* are more precisely described in the example flow for the RuuviTags, while others are not shown in use. The use of other nodes is following.

The *dht-sensor* and *ds18b20-sensor* are used to read the humidity and temperature sensors. The former node requires a node per sensor attached, while the latter can read all DS18B20 sensors attached. The *hostip* is simply used to get the current IP address; this could also be done with *hostname -I* command.

The node for Node-RED's Dashboard, *dashboard*, is used to provide a simple UI for user to access the most current data from the sensors. This UI is not used in large scale, as the one in Grafana is more visually attractive. The Node-RED UI can still be used, when no Internet connection is available, and no data is available from the InfluxDB and Grafana.

The *openweathermap* gathers weather data from OpenWeatherMap's API, with API key implemented into node. This weather data is sent over MQTT to Home Assistant to be viewed from UI or used by Snips AI, when enquiring for weather forecast. The *smooth* node is used to round-up numbers, mostly used for Node-RED's dashboard view.

While the other nodes are easy to install, the RuuviTag node is a bit more complex. The RuuviTag node's functioning depends on the *noble* node installed correctly; this can be installed from the *Manage palette*, as seen in the previous table.

To install RuuviTag node, the repository from Github needs to be cloned. Repository can be cloned to own home directory. (Jousimaa 2017)

```
# Clone repository in ~/git-clones/
$ git clone https://github.com/ojousima/node-red.git
# In folder with the package.json
$ sudo npm link
# In ~/.node-red directory
$ npm link node-red-contrib-ruuvitag
```

The *link* command links the repository and the needed packages to other Node-RED packages located in the *~/.node-red/node_modules* and the RuuviTag node appears, if correctly installed, in this directory as any other node would.

The BLE scan on the Raspberry Pi 3 may sometimes fail and refer to not having the permissions to start BLE scanning, with the following error message seen in the Node-RED's debug tab:

'Unable to start BLE scan. Adapter state: Unauthorized'

This error occurs because of the root privileges missing from the *node* binary, making it unable to start or stop the BLE scan. The following command gives the root/sudo privileges to the *node* binary, after Raspberry Pi reboots. (Noble – BLE central module 2015)

```
$ sudo setcap cap_net_raw+eip $(eval readlink -f 'which node')
```

The previous command assumes the *setcap* is installed. If not, it can be installed as an *apt* package: *sudo apt-get install libcap2-bin*.

7.4.3 Configuring Node-RED

After restarting the Node-RED service, the installed nodes are available for use from the list on the Node-RED editor and the flows can be created. To enable the Node-RED service to start upon the Raspberry Pi start-up, the service is enabled in *systemctl* :

```
$ sudo systemctl enable nodered.service
```

In some cases, the Node-RED may run out of memory and due to this, some actions may fail. The default value for allocated memory before Node-RED is ordered to clean up space is 128MB. On Raspberry Pi 3 and especially if no other high memory services are running, this value can be upped to 256MB to give Node-RED more space to operate.

```
# Edit /lib/systemd/system/nodered.service
...
Nice=5
Environment="PI_NODE_OPTIONS=--max_old_space_size=256"
...
# Reload configuration
$ sudo systemctl daemon-reload
```

After restart, the Node-RED is allowed to use up more space if needed. The default of 128MB should suffice in smaller flows but during the thesis work Node-RED crashed several times but, after increasing the allocated memory, remained stable.

Node-RED is protected with user credentials. These credentials are set in the *.node-red/settings.js* under the *adminAuth:* section. The username can be inserted here as plain text, but the password is generated with *node-red-admin hash-pw* command, which, if not installed, can be installed globally with the *npm*.

```
$ npm install -g node-red-admin
$ node-red-admin hash-pw
```

This asks the user for the password and will generate the hashed password to be used in the *settings.js* file below. After service restart, the Node-RED editor is accessed with the credentials made.

```
# Edit .node-red/settings.js
...
adminAuth: {
  type: "credentials",
  users: [{
    username: "<username>",
    password: "<generated_password_hash>",
    permissions: "*"
  }]
},
...
```

A problem noticed during the use of Node-RED was that the flows may stop working under certain circumstances, for example, when the Raspberry Pi is not used for a longer period of time and the Bluetooth module, BLE used by RuuviTags, may enter sleep mode. Normally re-deploying the flows works in this situation, however this requires modifications to be made to the flows, moving the nodes or adding new nodes to the flow. As this method is unsuitable for an independent system, a *curl* command can be used to automatically re-deploy the flows without interacting with them.

As the Node-RED is protected with credentials the *curl POST* needs to be authorised as well. An access token is needed for the command and it can be generated with the following command:

```
$ curl http://localhost:1880/auth/token --data 'client_id=node-
red-
admin&grant_type=password&scope=*&username=<username>&password=
<password>'
```

This generates the access token, which can be used with the *curl* command as followed.

```
$ curl -H "Authorization: Bearer <access_token>" -X POST
http://localhost:1880/flows -H "Content-Type: application/json"
--data @flows_<flowname>.json
```

The flowname is the name of the flows created, in this case *flows_sauna.json*, and is found in the *.node-red/* directory. The *curl* command itself is executed as a shell script, used with the *stoptimer* node (See Table 3) to run the script if RuuviTags have not been sending data for 10 minutes. The *stoptimer* resets whenever new message is received, continuously counting down until no data is received.

7.4.4 Building flows

The flows are built from the editor by dragging the nodes from the left panel to the canvas in the middle. The desired nodes can be connected together by wiring them either from the left or right side of the node. Once the wanted nodes are in place and the configuration if separately needed, is done, the now constructed flow can be deployed from the *Deploy* button at the top. By default, the deployment deploys all of the flows in active workspace, meaning all of the tabs as well, if multiple tabs for flows were created.

Since the flows created for the solution are large and the produced JSON code would be immense to present on thesis work, the full flows can be found on the project's Git repository (Saunaxio – NodeRED flow 2018). As an example of flow building, the flow for RuuviTags is shown, with the data sent to the MQTT broker for Home Assistant, and to the InfluxDB in DigitalOcean's machine (Figure 29).

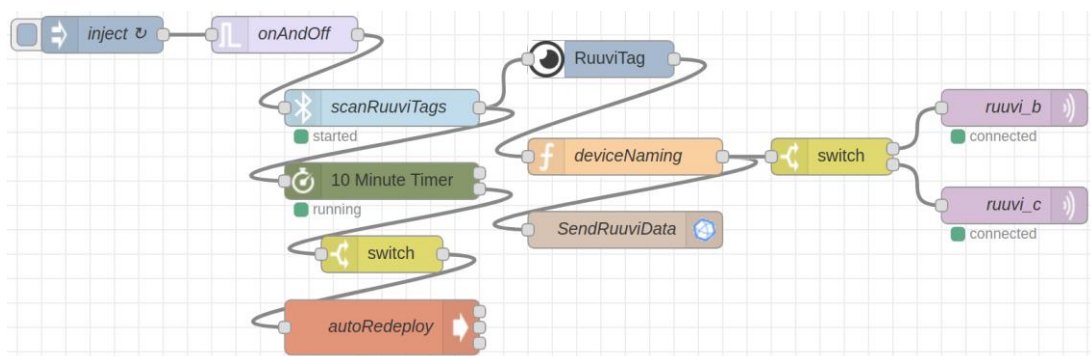


Figure 29. Flow to handle RuuviTag data

The flow for reading the RuuviTag is easy to accomplish in a manner shown in the figure (See Figure 29). The flow starts, much like any other flow, with an *inject* node, at far left. The *inject* is a timer, which sends a signal at specific intervals, at this point every 30 seconds. The signal is sent to the *trigger* node, which sends a signal *scan: true* to the *noble*, a BLE scanner, node to activate the scan sequence (Figure 30). The *trigger* delays the next message for a period of time and afterwards sends *scan: false* to stop BLE scanner. This method must be used in order to the BLE scanner to work, continuous scanning will have a negative effect on the Bluetooth module and at some point BLE scanner will jam and the Bluetooth speaker with it.

The image shows two configuration windows from the Node-RED interface. The 'Edit inject node' window on the left is configured with a payload of `{ "scan": false }`, a topic field, and a repeat interval of 30 seconds. The 'Edit trigger node' window on the right is configured with a 'Send' action of `{ "scan": true }`, followed by a 'wait for' delay of 5 seconds, and then a 'send' action of `{ "scan": false }`. It also includes options for resetting the trigger and handling the message.

Edit inject node

node properties

Payload: `{ "scan": false }`

Topic:

☐ Inject once after 0.1 seconds, then

Repeat: interval

every 30 seconds

Name: inject

Edit trigger node

node properties

Send: `{ "scan": true }`

then: wait for

5 Seconds

☐ extend delay if new message arrives

then send: `{ "scan": false }`

Reset the trigger if:

- ☐ msg.reset is set
- ☐ msg.payload equals optional

Handling: all messages

Name: onAndOff

Figure 30. Inject and trigger nodes

The *noble* node scans the surroundings for BLE devices, finding every Bluetooth device in the vicinity. The scan result is sent to the *RuuviTag* node, which parses the data specifically for RuuviTag entries. The RuuviTags send temperature, humidity, air pressure, battery voltage and acceleration data to the *msg.payload*, which can be easily used in the future (Figure 31). By default, RuuviTag entries are identified by only the *peripheralUuid*, which is the RuuviTags MAC address. The devices can be named if desired but for easier adding of new RuuviTags it is wiser to handle the devices with just MAC addresses.

```

msg : Object
  ▾ object
    payload: {"humidity":21.5,"temperature":26.94,"pressure":100424,"accelerationX":-76,"acc
    peripheralUuid: " "
    detectedAt: 1526473678904
    detectedBy: "sauna"
    ▸ advertisement: object
    rssi: -50
    _msgid: "71690bf8.07ac54"

```

Figure 31. Message from the RuuviTag node

The RuuviTags in use are specified in the *deviceNaming* function. Functions in Node-RED are written in Javascript, which allows basically any modifications to be made if deemed necessary. In this case the *peripheralUuid* is set as *msg.payload.topic* for easier database use in InfluxDB. Only the RuuviTags specified in the function are allowed to send data to the MQTT and the database (Figure 32).

Edit function node

Delete
Cancel
Done

node properties

Name

deviceNaming

Function

```

1 if (msg.peripheralUuid == " " ||
2 msg.peripheralUuid == " ")
3 {
4     msg.payload=JSON.parse(msg.payload);
5     msg.payload.topic = msg.peripheralUuid;
6     msg.topic = msg.payload.topic;
7     return msg;
8 }

```

Figure 32. Function node for message modification

The *if* statement in the function can also be executed with a *switch* node, where the confirmed and allowed *peripheralUuids* are set as following (Figure 33). The *switch* is placed and connected before the function, with both of the outputs connected to the *function* node.

Edit switch node

Delete
Cancel
Done

▼ node properties

Name

allowedMACs

Property

▼ msg. peripheralUuid

≡

==

▼

▼ a_z

→ 1

✕

≡

==

▼

▼ a_z

→ 2

✕

Figure 33. Switch used to sort out RuuviTags

After specifying the devices, the data can be sent to the InfluxDB (See Chapter 7.7.1). The *influx* output node requires specification of the database. New database connection is added to the node, with address and port of the InfluxDB service. The database name, along with the credentials to said database are given to the node (Figure 34). If the SSL/TLS connection is active in InfluxDB, the *certificate* and *private key* must be configured to the node (See Chapter 7.7.1) in *.pem* format, as Node-RED only supports this format. The certificates can either be uploaded to the Node-RED or used from local files.

Edit influxdb out node > **Edit influxdb node**

Delete
Cancel
Update

Host
Port
8086

Database

Username

Password

☒ Enable secure (SSL/TLS) connection

TLS Configuration

Name

Figure 34. Creating InfluxDB node connection

While the InfluxDB node handles the data sending to the database, having the RuuviTags to show the most current data in Home Assistant without a new database connection is achieved with MQTT.

MQTT nodes, both input and output, are in-built with the Node-RED installation. The setup uses the output node to send current data to Home Assistant, with each sensor to own topic (Figure 35). With the MQTT broker not being protected, as configured to only function on localhost, the setup consists of only server address and port number, which are added to the node's broker connection.

The message payload is sent through a switch, which ensures that correct RuuviTag sends data to correct MQTT broker topic. The validation is done with the help of *peripheralUuid*, as described before.

The *topic* is subscribed to in the Home Assistant to get the latest data to show on the Home Assistant's UI and for the Snips AI to use. This is specified in Home Assistant's and Snips AI's chapters (See Chapters 7.6.2 and 7.10.1).

Edit mqtt out node

Delete
Cancel
Done

▼ node properties

Server
localhost:1883

Topic
sensors/ruuvitag/ruuvi_b

QoS

Retain

Name
ruuvi_b

Figure 35. Node-RED's MQTT broker node

As explained before, the Node-RED's flows may stop from working, if the device is not used for a longer period of time. This is especially noticed with the *noble* node and the BLE scanning, resulting in a failure with Bluetooth devices.

The *stoptimer* node was placed to counter the problem. The *stoptimer* is set to a certain amount of time, in this case 10 minutes. With each BLE scan from the *noble* node, the timer resets and restarts the countdown process. Should the BLE scan fail, the *stoptimer* reaches the end and activates the following nodes. The *switch* between the *stoptimer* and the *exec* node (Figure 36) ensures the message payload of "1" passes through to the system command in *exec* node. The system command runs the *curl* command specified earlier (See Chapter 7.4.3).

Edit exec node

Delete Cancel Done

▼ **node properties**

Command

+ Append ☐ msg.payload

➡ Output

Figure 36. Exec node for system commands

Same flow method is used around Node-RED with sensor related flows. These other flows are not shown on the thesis writing; however, the final flows are available in project's Git repository (See link at start of Chapter 7.4.4).

7.5 Mosquitto

Mosquitto MQTT broker is used by Home Assistant and Snips AI, former to show the current data from Node-RED and the latter to send messages, intents, and to communicate between different services of Snips AI.

The broker service is provided by Eclipse's Mosquitto. Mosquitto is installed on the Raspberry Pi and can be installed as an *apt* package, both Mosquitto and Mosquitto clients command line tools.

```
$ sudo apt-get install mosquitto mosquitto-clients
```

By default, the MQTT broker is not protected with user credentials. Because of this, the broker can be accessed by anyone from anywhere inside the local network. This can be problematic, as the network may also be available to larger audience, and MQTT open for attacks and foul use. Since both services, the Home Assistant and the Snips AI, use the same MQTT broker to transfer messages, protecting the broker comes difficult. Home Assistant's configuration for the MQTT broker supports the user credentials, both username and password. This is

not the case with the Snips AI. At the time of writing the user credentials cannot be implemented into Snips' services and protecting the MQTT broker with credentials results in a failure, when starting the Snips AI services.

An alternative way of protecting the MQTT broker is to have it configured to function only on *localhost*. This allows only services running on the localhost to access the broker, while connecting from remote machine, when in the same network, results in a failure.

```
# Edit /etc/mosquitto/mosquitto.conf
...
bind_address localhost
```

The MQTT broker can be tested by sending a test message to the broker, while reading the topic at the same time. Over the SSH, the possibility of having two terminals open is the easiest way.

```
# Subscribe to test topic
$ mosquitto_sub -h localhost -p 1883 -t 'test'
# On another terminal, publish to topic test
$ mosquitto_pub -h localhost -p 1883 -t 'test' -m 'Hello'
```

The message 'Hello' is published to the broker topic *test* and can be seen when subscribing to the topic. The same method is used by the Home Assistant and Snips AI. Node-RED publishes a message, containing the measured data in JSON format, to a certain topic, while the Home Assistant subscribes to said topic and presents the data in more readable form in the UI. Snips AI uses the MQTT to communicate between Snips' services and handle the intents executing actions according to user inputs.

The Node-RED's MQTT node uses the topics similarly to the example above, each sensor publishes the message to a certain topic, to which the Home Assistant subscribes to get the latest data to the UI and for the Snips AI. The MQTT node is explained in Node-RED's chapter (See Chapter 7.4.4)

At some point during the development the MQTT service entered failed state, trying to re-establish the connection to the broker. This resulted in Node-RED's

MQTT nodes to lose connection and dropping the data for Home Assistant. Snips AI, being dependent of the MQTT broker, also failed. The Mosquitto trying to establish the broker took about 100% of the CPU's power, breaking almost every service on the Raspberry Pi. This problem may result from the lost Internet connection to the router's WiFi module, as described earlier (See Chapter 7.2.2), during the period of time with wired connection to another network, while the router is active. Restarting the Mosquitto service and / or rebooting the Raspberry Pi, as well as making sure the Internet connection is available, should fix the problem.

7.6 Home Assistant

Home Assistant is pre-installed on the Hassbian image and is usable and accessible from the start, at *hassbian.local:8123* by default but can be changed to another hostname if necessary (See Chapter 7.2.2).

Home Assistant uses own user to run the service, by default *homeassistant*. The configuration files of the Home Assistant are all located under *homeassistant* user's home directory at */home/homeassistant/.homeassistant/*. The configuration files use *.yaml* data serialisation language. By default, the *configuration.yaml* is the file containing all the needed configuration for the Home Assistant.

As all the configuration will be in the same file, the configuration may grow exceedingly large and hard to read and maintain. The Home Assistant supports the dividing of the configuration to separate files, under the same directory that can then be referenced in the main configuration file, the *configuration.yaml*. For clarity of the configurations, the sensors, the relays and the intent scripts for Snips AI, are in separate files amongst others.

Home Assistant also supports a more secure way of handling the confidential data, such as tokens, passwords and keys. While the text is still in plain text, the data can be stored into separate file, the *secrets.yaml*, which can be ignored, for example, when pushing the files to Github. This way the configuration files of

different components can be shared for others to use, while the confidential data of tokens, passwords and keys is kept apart.

The Home Assistant is protected with a password for the UI on the browser. This password is defined in the *secrets.yaml* in plain text. As the Home Assistant is by default only accessible from the local network, the protection of the network itself is important. When exposing the Home Assistant to a larger network, through a DuckDNS for example, other safety measures should be taken into consideration. In this setup, the base level protection for the Home Assistant will suffice.

7.6.1 Configuration

The main configuration file is the *configuration.yaml* located in the */home/homeassistant/.homeassistant/* directory. Other configuration files created separately for each component used in the Home Assistant are similarly located and are used by including them in the *configuration.yaml* file.

```
# In configuration.yaml
homeassistant:

...
# The secrets.yaml is referenced for password
http:
  api_password: !secret api_password
...
group: !include group.yaml
switch: !include switch.yaml
intent_script: !include intent_script.yaml
...
```

With the Hassbian installation the Home Assistant is enabled to start on Raspberry Pi's boot. No configuration is needed for the Home Assistant service itself, however, the configuration files for the components must be created accordingly. The full configuration files used in the Home Assistant can be seen from the projects Git repository. (Saunaxio – Home Assistant 2018)

While executing *shell_commands* inside the Home Assistant, some may fail due to permission problems. This is corrected by adding *homeassistant* user into *sudoers*, thus giving permission to execute shell commands as *sudo* without continuous password prompts, when *sudo* is needed.

```
# Add homeassistant to sudoers
$ sudo visudo
...
homeassistant ALL=(ALL) NOPASSWD: ALL
...
```

7.6.2 Adding components

While using the Hass.io image, Home Assistant's official image for devices such as Raspberry Pi, the components are added from within the browser UI. As this image is not used, in Hassbian the components are added from the configuration files by either adding the component entry to the main configuration file, the *configuration.yaml*, or by creating new *.yaml* file for the component. For example, adding of the Time & Date and MQTT sensor components are described.

The available components for the Home Assistant can be seen from the *Components* section on their homepage (Home Assistant – Components 2018). By searching for the wanted component, the arguments for said component are described, with example use cases as well. To add the Time & Date component, the needed configuration is added to the *sensor.yaml* configuration file.

```
# New file sensor.yaml
- platform: time_date
  display_options:
    - 'time'
    - 'date'

# In configuration.yaml include the new .yaml file
...
sensor: !include sensor.yaml
...
```

The Home Assistant supports some actions through the browser UI. The configuration file can be checked for possible errors or problems under the

Settings and *Configuration* section of the UI. If the configuration created before passed, the service can be restarted from within the UI. After the restart, the newly added component should be visible in the *Overview* tab (Figure 37).

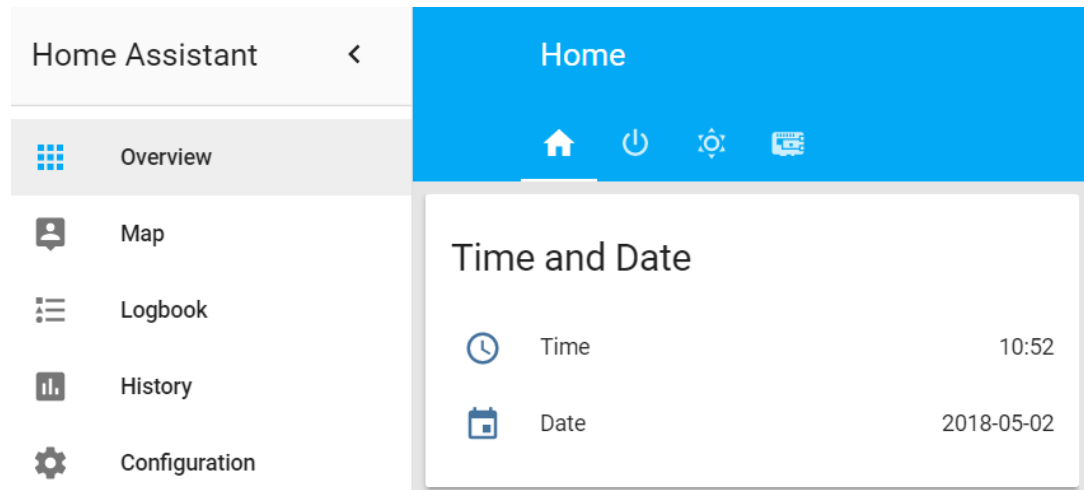


Figure 37. *Time&Date* component in Home Assistant

If the instruction of above were followed, the view should be different from the above, as by default the *sensor* components are presented in a circular form.

This form works, when presenting temperature, time or status of some service. When adding multiple sensors, however, the view will get hard to read. Therefore, the Home Assistant offers customisation for the components to be viewed as shown in the figure above. The components can be divided into separate *groups* in separate *views* (See Chapter 7.6.3).

As Home Assistant is used to present the data sent by the Node-RED via MQTT, the main component used in Home Assistant is the MQTT sensor. The MQTT sensor subscribes to the *state_topic* of the MQTT message, e.g. *sensors/ruuvitag/ruuvi_b* created in the Node-RED MQTT node, and shows the data with *value_template* in the Home Assistant's *group*. The configuration example of such MQTT sensor is following.

```
# Edit sensor.yaml file
...
# RuuviTag_1
- platform: mqtt
  state_topic: 'sensors/ruuvitag/ruuvi_b'
```

```

    name: 'ruuvi_air_temp'
    unit_of_measurement: ' °C'
    value_template: '{{ value_json.temperature }}'
...

```

Same configuration model is used for all MQTT sensors, all of which are located in the same file. The MQTT sensors will appear in circular form at the top of the *Overview* tab. Similarly to the *Time & Date* component, the *group* is added to make the measurements easier to read.

The component needed by Snips AI is the *intent_script*, which is used to handle the *intents* created for the Snips skills in Snips Console (See Chapter 7.10.2). The intent scripts are located in a separate *.yaml* file and are included to the main *configuration.yaml* file like any other component. Home Assistant's Snips AI component is activated with *snips:* in the *configuration.yaml*.

The configuration file for the *intent_scripts* is large, with every intent created for the Snips skill having own actions when executed. As an example of the file construction, the handling of a Snips skill for lights is described. The skill used is a ready-made bundle from the store, forked under own user for possible modifications.

```

# configuration.yaml
...
snips:
  feedback_sounds: true
  intent_script: !include intent_scripts.yaml
...
# intent_scripts.yaml
...
lightsTurnOnSet:
  action:
    service: switch.turn_on
    data_template:
      entity_id: switch.light_1, switch.light_2

LightsTurnOff:
  action:
    service: switch.turn_off

```

```

data_template:
  entity_id: switch.light_1, switch.light_2
...

```

The *intent* recognised by Snips AI activates the *intent_script* component, executing the *service* for entities in *data_template*. If the user desires to be notified on such actions, the *shell_command* can be used to send audio notifications with *puhu*. The *shell_commands* are configured in an own file, in a same way the *intent_scripts* are.

```

# shell_commands.yaml
puhu: puhu {{ puhu }}
# Use in the lightsTurnOnSet example
...
lightsTurnOnSet:
  action:
    - service: switch.turn_on
      data_template:
        entity_id: switch.light_1, switch.light_2
    - service: shell_command.puhu
      data_template:
        puhu: "Valot päällä"
...

```

The *puhu* notifications are used for an intent *searchWeatherForecast* from Snips' bundle. The weather forecast is pulled in Node-RED from *openweathermap* node and sent to MQTT broker to be used in Home Assistant. The *intent_script* speaks the weather forecast out loud, updating from OpenWeather API approximately every 30 minutes. The API requires an API key, which is free and acquired from OpenWeatherMap's webpage after registration. (OpenWeatherMap – API)

In case of a failure in one or more services running on the Raspberry Pi, the *script* component could be used to restart said services from within the Home Assistant UI. Such additions were, e.g. Node-RED, Snips AI, Mopidy and networking services, the latter in case of force restart, if the configuration for the Internet access was modified. Each *script* executed a *shell_command*, with the executable script on the Raspberry Pi. The *script* component was added to the *group.yaml* in order to view it on the UI panel (Figure 38).

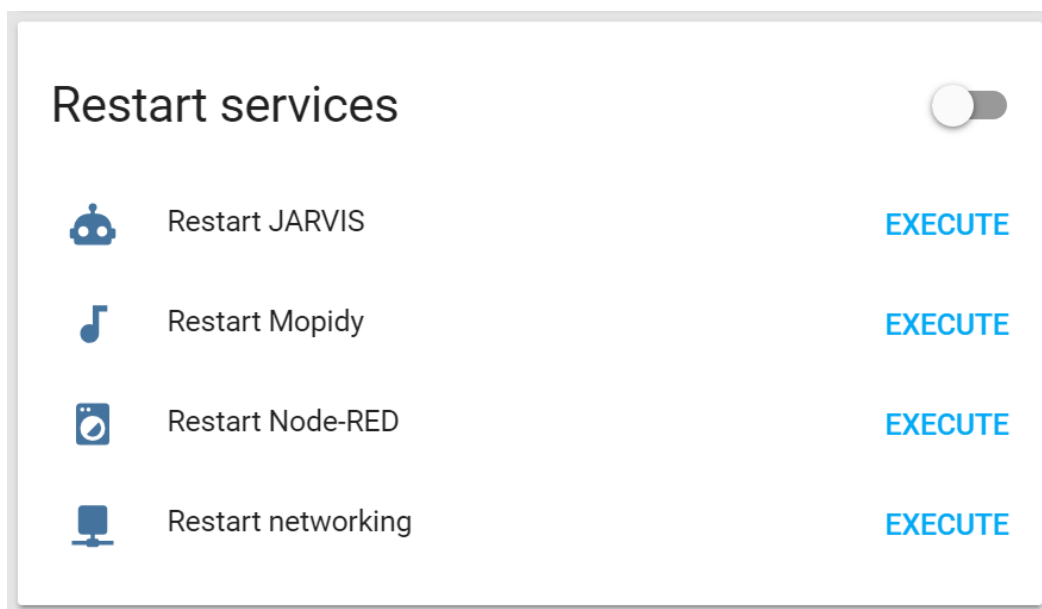


Figure 38. Executable scripts in Home Assistant

7.6.3 Customisation

Adding the components into separate groups improves the reading and using of the components. The *group* configuration is added into own *group.yaml* configuration file, located in the same directory with other similar files, and included into the *configuration.yaml* file as previously described with the *sensor.yaml* file. The following configuration shows the two described components in separate groups in *Overview's Home* tab.

```
# New file group.yaml
default_view:
  view: yes
  icon: mdi:home
  entities:
    - group.timedate
    - group.ruuvitag_1

timedate:
  name: Time and Date
  entities:
    - sensor.time
    - sensor.date
```

```

ruuvitag_1:
  name: Incoming air
  entities:
    - sensor.ruuvi_air_temp

```

The naming of the *groups* and *entities* must be noted, if done falsely *group* will not show up on the *default_view*, or *Home*, tab (Figure 39).

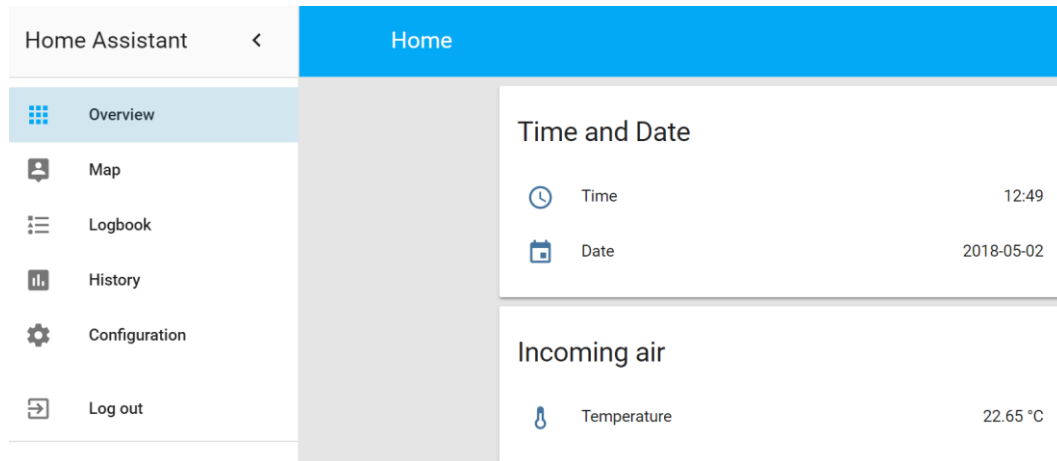


Figure 39. Group view of Home Assistant

While most of the customisation done for the Home Assistant in thesis work is done in similar manners as above, the names of the sensors, such as in figure (See Figure 39) for the *Incoming air* temperature, as well as the icons are set in the Home Assistant's internal customisation section.

The customisation is accessed from the left-side panel under the *Configuration* and *Customization*. The customisation can be done to either groups or components, in this case for the *Incoming air* RuuviTag sensor measurement (Figure 40).

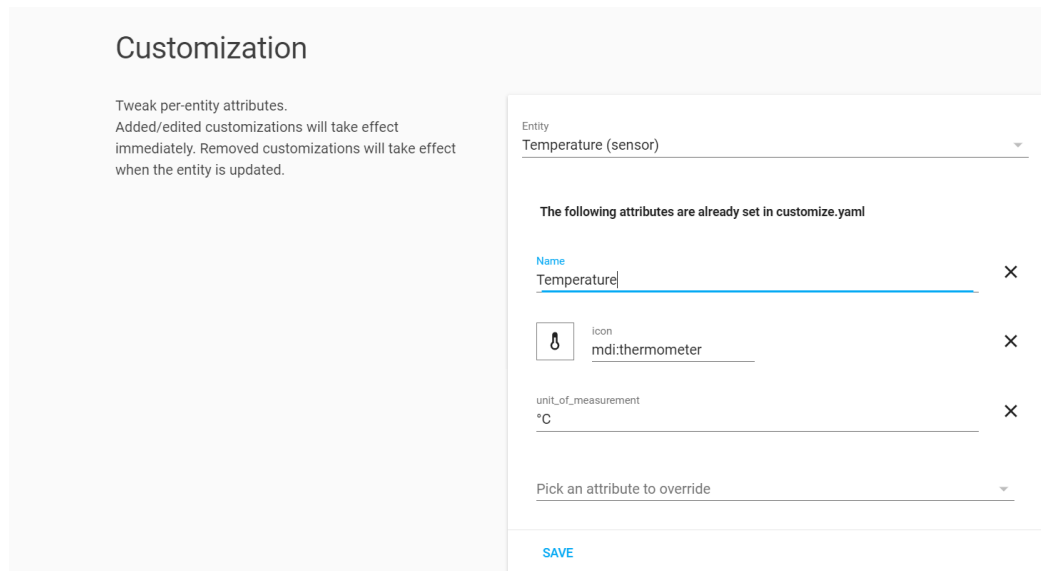


Figure 40. Customisation view of Home Assistant

The Home Assistant supports the use of MDI (Material Design Icon) icons. Icons can be found from MDI's webpage and any icon can be used from a vast variety for specific purposes. (Material Design Icons)

Optionally, a *state_card_custom_ui* can be added to provide more customisation options. These state cards are installed manually to the Home Assistant's directory and can be retrieved from Github. (Custom UI – Home Assistant 2018)

The custom UI was not used in large scale but was used for the switches, the lights and water pump, to show the time since last active or activated on the frontend UI, as this functionality is not supported by default in Home Assistant (Figure 41).

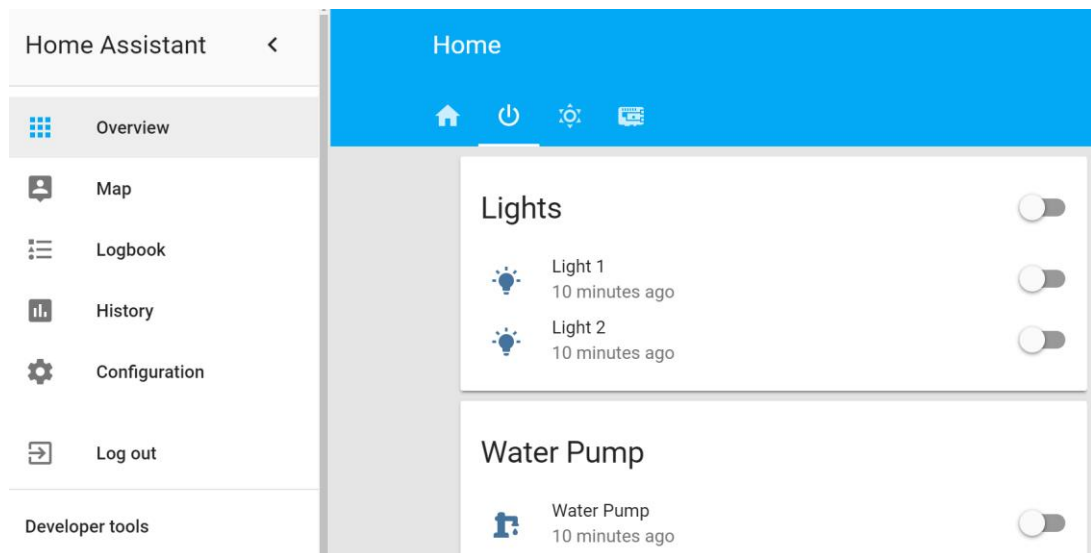


Figure 41. Custom UI showing *last_active* in the Home Assistant

The custom UI installed consists of two *.html* files and *.gz* packets, which are set in the newly created directory *www/custom_ui/*, contents being following.

```
$ ls /home/homeassistant/.homeassistant/www/custom_ui/
state-card-custom-ui-es5.html    state-card-custom-ui.html
state-card-custom-ui-es5.html.gz state-card-custom-ui.html.gz
```

By installing the custom UI manually, updated versions must be installed manually the same way as above. The custom UI is activated in the *configuration.yaml* as follows.

```
# Edit configuration.yaml
...
homeassistant:
  customize_glob:
    switch.*:
      custom_ui_state_card: state-card-custom-ui

frontend:
  javascript_version: latest
  extra_html_url:
    - /local/custom_ui/state-card-custom-ui.html
  extra_html_url_es5:
    - /local/custom_ui/state-card-custom-ui-es5.html
...
```

In this case, to get the *last_active* object to show, it must be activated from the *Cuscomization* section, from the drop-down of the specific switch. Home Assistant updates the object reading every 30 seconds by default but may fail to do so, in which case the *last_active* may show false info and refresh of the UI page is needed.

7.6.4 Automation

The automation for some functionalities, e.g. the automatic water pump deactivation and determining, whether the sauna is being stolen or not, are done in the Home Assistant's in-built *Automation* section. The automation can also be done directly to the *automations.yaml* in Home Assistant's directory, however, the UI is significantly easier to modify. Created automation will appear in the *automations.yaml* nonetheless, and may be modified there also.

The idea of automation in Home Assistant is to have a component respond as instructed, when certain conditions are fulfilled. This process is started by a *trigger*.

Triggers are based on actions or statuses of components, such as when certain humidity limit is exceeded, or when a switch is switched on. The *trigger* executes the *action* either directly or when certain *conditions* are met. Conditions are optional and may be used to restrict the triggering from happening.

Such conditions may be e.g. the corresponding temperature measurement being too high for the triggering humidity reading, or that another switch is already switched on. When the conditions are fulfilled, the automation moves to last section, the *action*.

Actions are executed by the trigger, after possible conditions are met. When humidity drops below the set limit, and the current temperature is within the set conditions, the action of TTS activates, stating that more water should be applied on the stove. Similarly, the water pump could be activated to automatically pump water on the stove.

The automation is used to automatically switch off the water pump after activated with either voice command or manually from Home Assistant's UI. The automation is *triggered*, when the *state* of the *switch.water_pump* changes from *off* to *on* (Figure 42)

Triggers

Triggers are what starts the processing of an automation rule. It is possible to specify multiple triggers for the same rule. Once a trigger starts, Home Assistant will validate the conditions, if any, and call the action.

[Learn more about triggers.](#)

Trigger type
State

Entity
switch.water_pump

From
off

To
on

ADD TRIGGER

Figure 42. Trigger for automation

The *action* is triggered afterwards. The *action* contains a delay of 3 seconds, which can be changed according to the water pump's power and the actual water amount pumped from the tank. After the delay, the *switch.water_pump* is switched back of, the *state* returns to *off* (Figure 43).

Actions

The actions are what Home Assistant will do when the automation is triggered.

[Learn more about actions.](#)

Action type
Delay

Delay
00:00:03

Action type
Call service

Service
switch.turn_off

Service data
{
 "entity_id": "switch.water_pump"
}

Figure 43. Action for automation

The automation described before can be executed with another automation based on the current temperature and humidity. These could either be hardcoded into the automation, or set with an *input_number*, a slider, which can be adjusted to preferred temperature and humidity levels. The *puhu* command could be used to

notify the user when applying water onto stove is appropriate, according to the set levels with the sliders. Similarly, the system could notify when preferred temperature is reached, or when a break would be in order.

7.7 Cloud services

To start gathering the data from the Node-RED the cloud services must be installed and configured. As the cloud infrastructure is provided by the DigitalOcean, the configuration needed for the cloud infrastructure itself is minimal. The used machine is not the most powerful one but presents enough resources to handle the needed services with ease.

The server machine can be rented from the DigitalOcean's homepages after providing the necessary billing information. The machine used had 1GB of RAM, 2.4GHz processor and about 25GB of storage, costing at around \$5 per month at the time of writing in April 2018. Operating system in this case was Linux Ubuntu 16.04, according to the machines info (DigitalOcean – Pricing 2018.)

After the information for the DigitalOcean is in order, the specifications of the rented machine are received via the specified route. The most important part of this information is the address and the password for the machine. The server machine can be accessed via SSH connection, after which the password should be changed into more secure one. After password changing the server is ready for other service's installation.

7.7.1 Influx database

The database is installed on the DigitalOcean machine. The installation of the InfluxDB is done with few quick steps, which install the latest stable version of the InfluxDB, version v.1.5.1 at the time of writing, for Linux Ubuntu 16.04. The following installation method is valid at the time of writing.

```
$ curl -sL https://repos.influxdata.com/influxdb.key | sudo  
apt-key add -  
$ source /etc/lsb-release
```

```
$ echo "deb https://repos.influxdata.com/${DISTRIB_ID,,}
${DISTRIB_CODENAME} stable" | sudo tee
/etc/apt/sources.list.d/influxdb.list
$ sudo apt-get update && sudo apt-get install influxdb
$ sudo service influxdb start
```

After the database installation, configuration needs to be done properly to secure the database from free access. The InfluxDB uses a configuration file located in */etc/influxdb/influxdb.conf*, where modifications can be made accordingly.

For the safe use of the database, an SSL (Secured Sockets Layer), or TLS (Transport Layer Security) certificates are added to enable the HTTPS connection to the database and to authorize actions to and from the database. The SSL certificates are created with *openssl* command found on many machines by default. The InfluxDB mainly supports two types of certificates, either the CA-signed certificates, issued by the certificate authority, or self-signed certificates, which are created on the local machine. The latter method was used to create two files, the private key and a self-signed certificate, with the following command.

```
$ sudo openssl req -x509 -nodes -newkey rsa:2048 -keyout
private.pem -out certificate.pem -days <number_of_days>
```

The *openssl* command creates the self-signed certificate, *private.pem*, and private key, *certificate.pem*, with the *-days* as the number of days before the certificate expires. The number can be up to around 11 000 days, which is about 30 years, to prevent key from expiring for quite some time. Larger numbers can break the SSL certificate, resulting in a failure during the command.

Both files are moved to the */etc/ssl/influx/* directory, which, if not present already, is created for this purpose. The certificate and the key both needs to be in *.pem* format, as the InfluxDB node used in the Node-RED can only use these files, according to the description of said node.

To enable the HTTPS for InfluxDB the *influxdb.conf* configuration file in */etc/influxdb/* is modified. No other modifications are needed than the ones shown below.

```
# Edit [http] section in /etc/influxdb/influxdb.conf
...
[http]
    auth-enabled = true
    https-enabled = true
    https-certificate = "/etc/ssl/influx/certificate.pem"
    https-private-key = "/etc/ssl/influx/private.pem"
...
```

The database needs to have an admin user and the database should be used with another user, with no admin privileges. The admin user is created from the InfluxDB CLI, with admin privileges, together with the normal user. Because of the SSL secured connection the Influx CLI is no longer accessible as previously but instead needs to be opened as below.

```
$ influx -ssl -unsafeSsl
# Admin user
> CREATE USER <username> WITH PASSWORD '<password>' WITH ALL
PRIVILEGES
# New non-admin user
> CREATE USER <username> WITH PASSWORD '<password>'
# Grant READ / WRITE privileges to <username>
> GRANT ALL PRIVILEGES TO 'username'
# Authenticate as admin user
> auth
username: <username>
password: <password>
> CREATE DATABASE <database_name>
# Move into database
> USE <database_name>
```

Creation of the measurements is done from the Node-RED when the first data is sent to the database. The measurement is named at the same time.

After the measurements are created inside the database, the InfluxDB CLI can be used to see the measurements, if needed. Otherwise, the need for the InfluxDB CLI use is not present, as the data queries for monitoring is done from the Grafana visualisation tool. The InfluxDB CLI can be used to quickly view the latest

data from a measurement inside the database, for example from RuuviTags (Figure 44).

```
> select temperature, humidity, pressure, battery, topic from ruuviTagDB limit 10;
name: ruuviTagDB
time          temperature humidity pressure battery topic
-----
1523619388639378073 23.54      11.5      102077    3013
1523619413828229585 23.14      11.5      101974    3031
1523619426635250948 23.54      11         102080    3007
1523619440926906741 23.15      11.5      101970    3031
1523619449864024638 23.14      11.5      101968    3013
1523619522614309393 23.53      11         102077    3001
1523619529874620581 23.14      11.5      101974    3019
1523619567844885681 23.15      11.5      101973    3013
1523619568639418718 23.53      11         102067    3001
1523619593874966361 23.15      11.5      101970    3019
>
```

Figure 44. Measurement for RuuviTags

7.7.2 Grafana install and configuration

Grafana is installed on the DigitalOcean machine together with the InfluxDB. The following installation is for the Ubuntu 16.04 and installs the latest stable version of Grafana, version 5.0.4, at the time of writing.

```
$ wget https://s3-us-west-2.amazonaws.com/grafana-releases/release/grafana_5.0.4_amd64.deb
$ sudo apt-get install -y adduser libfontconfig
$ sudo dpkg -i grafana_5.0.4_amd64.deb
# Start service
$ sudo systemctl daemon-reload
$ sudo systemctl start grafana-server
# To enable service start on system startup
$ sudo systemctl enable grafana-server.service
```

The Grafana is accessible on the server machine at port *3000* as default. The configuration for the database connection is done within the browser UI.

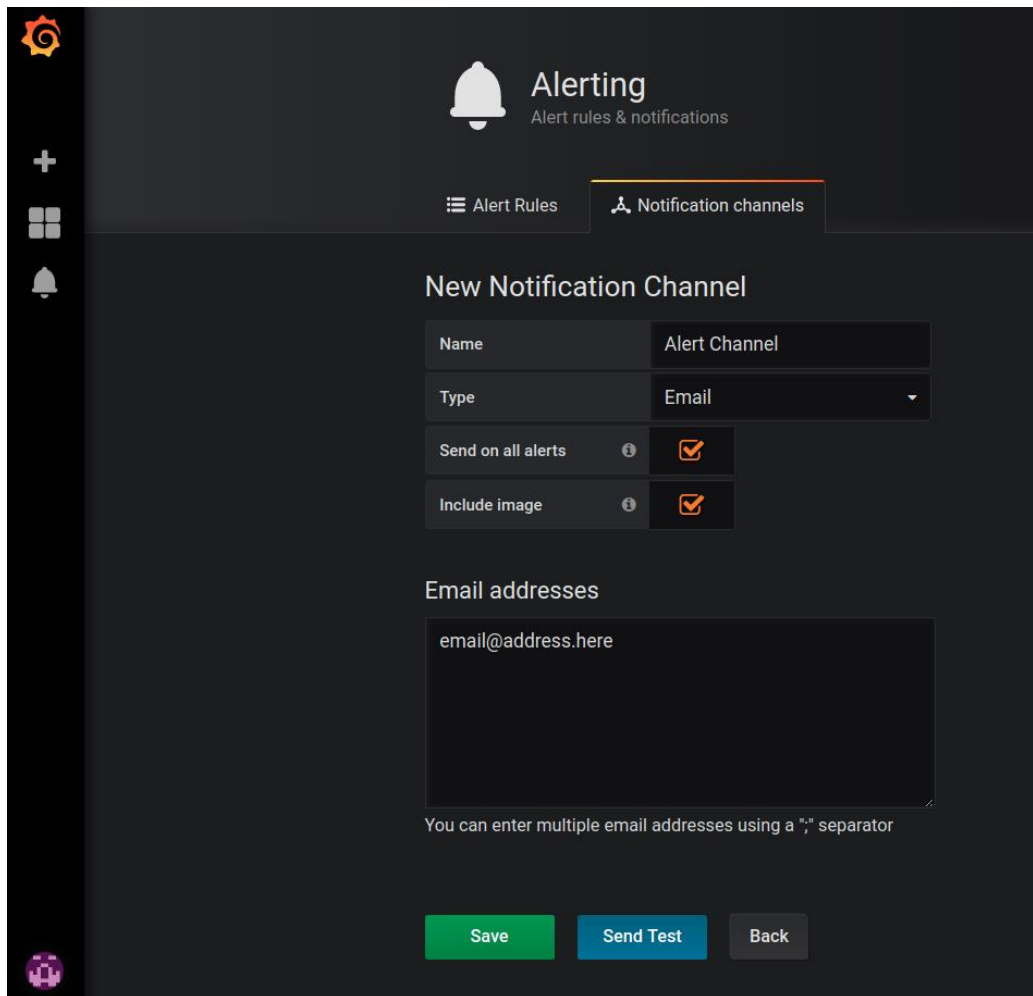
The system built can send email notification and alerts, based on certain boundaries set, to the user. This needs to be enabled in the configuration file for Grafana, in */etc/grafana/grafana.ini*. The email notification channel described supports the Google's Gmail email service, but other configurations are also available for preferred notification channels. (Grafana alert notifications 2017)

```
# Edit [smtp] section in /etc/grafana/grafana.ini
...
[smtp]
enabled = true
host = smtp.gmail.com:465
user = <user_email_address>
password = <device_password_from_gmail>
skip_verify = false
from_address = <notification_sender_address>
from_name = <notification_sender_name>
...
```

The password for the *smtp* service is made in Google Gmail by enabling the 2-Step verification and adding a device password for custom application. This is done in the Gmail's user settings, under the *Sign-in & Security* section and following the instructions (Google 2-Step verification 2018.)

After the 2-Step verification is functioning as intended the password can be created for a specific application. Under the *Security* and *Signing in to Google*, the app password is created, and the 16-digit code can be used as password in the Grafana configuration file in */etc/grafana/grafana.ini* (Google – App passwords 2018.)

Once the above steps are done correctly the notification channel of email type can be added under the *Alerting* and *Notification channels* (Figure 45). The alerts and notifications can be sent to any email address, regardless of the service provider. Multiple addresses can be added with a ';' separator.



The screenshot shows the Grafana Alerting interface. At the top, there's a header with the Grafana logo, a plus icon, a grid icon, and a bell icon. The main header area is titled 'Alerting' with the subtitle 'Alert rules & notifications'. Below this, there are two tabs: 'Alert Rules' and 'Notification channels'. The 'Notification channels' tab is active. The main content area is titled 'New Notification Channel'. It contains a form with the following fields: 'Name' (with the value 'Alert Channel'), 'Type' (with a dropdown menu showing 'Email'), 'Send on all alerts' (with a checked checkbox), and 'Include image' (with a checked checkbox). Below these fields is a section titled 'Email addresses' with a text input field containing 'email@address.here'. A note below the input field states: 'You can enter multiple email addresses using a ";" separator'. At the bottom of the form, there are three buttons: 'Save' (green), 'Send Test' (blue), and 'Back' (grey).

Figure 45. Adding a notification channel for Grafana

The *Alerts* for which the *Notification channel* is used are created under each graphs *Alert* tab, where specific boundaries for, e.g. temperature, can be set and notifications or alerts can be sent accordingly (Figure 46).

Alert Config

Notifications (1) **Name** RuuviTag Temperature Alert **Evaluate every** 10m

State history

Delete

Conditions

WHEN	max ()	OF	query (A, 5m, now)	IS ABOVE	85	
OR	max ()	OF	query (B, 5m, now)	IS ABOVE	85	
OR	min ()	OF	query (A, 5m, now)	IS BELOW	-30	
OR	min ()	OF	query (B, 5m, now)	IS BELOW	-30	
+						

If no data or all values are null **SET STATE TO** No Data ▼

If execution error or timeout **SET STATE TO** Alerting ▼

Figure 46. Alert rules for RuuviTags

7.7.3 Adding data source to Grafana

The InfluxDB is found in the Grafana by default and can be added as a data source. To add a data source the user must have *admin* privileges, as *editor* or *viewer* have no privileges to access the more delicate parts of Grafana service. The first user used to access the Grafana should have *admin* privileges.

The data source is added from the *Configuration* section on the left panel of the UI. The used InfluxDB is found by default from the list of available data sources. The data source is given a name, which can be used later when creating queries. By default, the URL for the database is *localhost*, as in this setup, as both services run on the same machine. The authentication settings can be chosen as seen fit, with the basic authentication of user credentials for the InfluxDB. The user used for accessing the InfluxDB should not be *admin* user of the database, since the non-admin user cannot accidentally drop all the measurements, thus deleting all the gathered data. The InfluxDB details are added at the end of the *Data Sources* page.

Once the data source is successfully added, the data can be shown on a graph or singlestat panel in a dashboard. This is done by making a new dashboard, if not

already made, and by adding a new graph to show the timestamped data on (Figure 47).

Graph General Metrics Axes Legend Display Alert Time range

Data Source ▼ ► Options

A FROM default ruuviTagDB WHERE topic = mac_address +

SELECT field (temperature) last () +

GROUP BY time (\$_interval) fill (null) +

FORMAT AS Time series ▼

ALIAS BY RuuviTag B

B SELECT last("temperature") FROM "ruuviTagDB" WHERE ("topic" = 'mac_address') AND \$timeFilter GROUP BY time(\$_interval) fill(null)

C Add Query

Figure 47. Creating graph for RuuviTags

The query of above generates a timeseries graph, which presents the data from specified time range on the top right corner of the UI. In this case, the graph was created to show the data from RuuviTags (Figure 48).

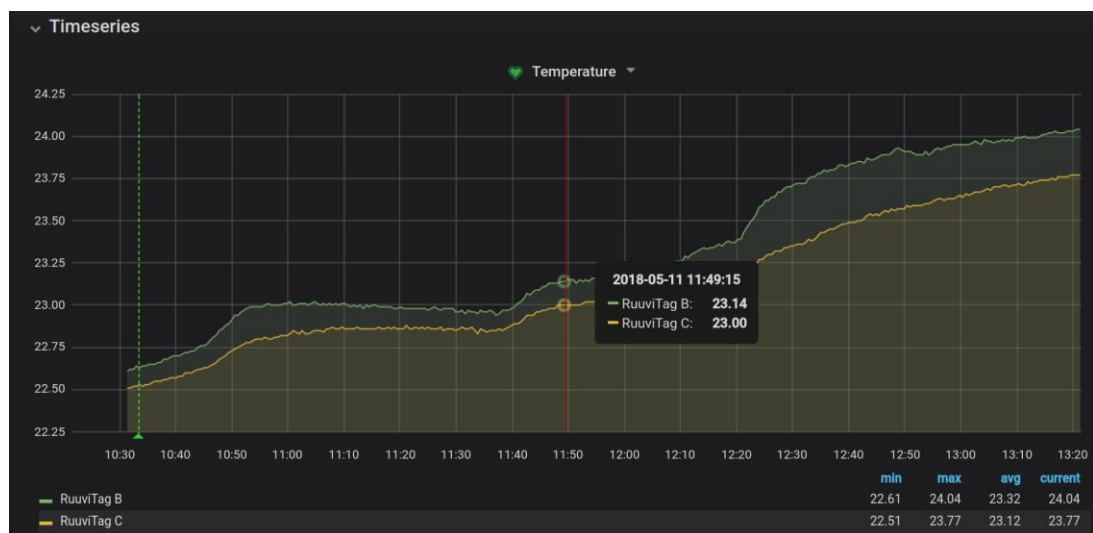


Figure 48. Graph for RuuviTag temperatures

Similarly, the data can also be presented as singlestat data, showing only the latest measurement or median of a number of latest measurements. In the setup a custom-made *D3 Gauge* plugin, found from the official Grafana plugin list, was used to show the singlestat data on (Figure 49).

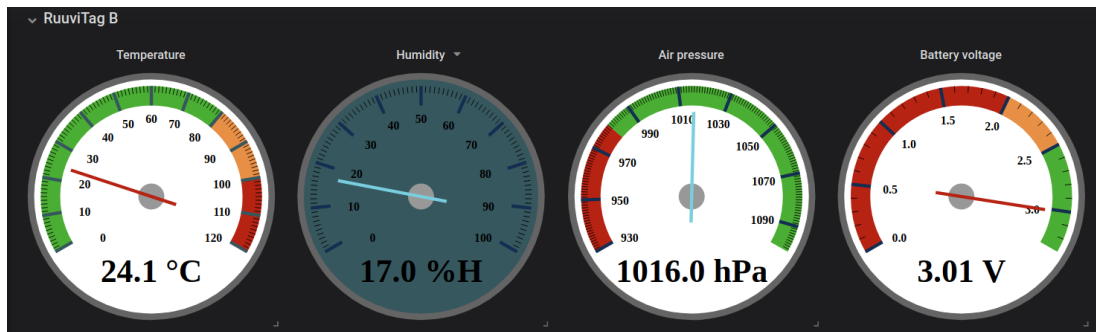


Figure 49. *D3 Gauges* used for RuuviTag singlestat

The singlestat gauges are customised similarly to the graphs, where the threshold colours can be changed accordingly and the size of different components, for example the fonts and labels. No larger instructions for the customisation of the Grafana dashboards is present in thesis, as the customisation is easily done by testing out different values and settings.

7.8 Audio system

The audio system includes the configuration of the UE Roll 2 Bluetooth speaker (SP1) and the USB microphone (MIC1) to be used in the solution, as well as the Text-to-Speech (TTS). Service specific configuration for audio use is handled in each service's section separately.

The audio output from the Raspberry Pi 3 can be done with either physical connection via the 3.5mm audio jack, provided the speaker has its own power source, as the Raspberry Pi can supply little to none power through the jack, or with a Bluetooth connection, which in comparison needs more configuration.

7.8.1 Connecting Bluetooth speaker

The Bluetooth connection between the Raspberry Pi 3 and a Bluetooth speaker is relatively easy if the GUI can be used. As this is not available on Hassbian, the connection needs to be done via terminal with a few commands. This works on Raspberry Pi 3 Model B and on its in-built Bluetooth module.

```
# Install bluetooth, bluez and bluealsa
$ sudo apt-get install bluetooth bluez bluealsa
```

```
# Pair bluetooth speaker
$ bluetoothctl
> scan on
> info <speaker_mac_address>
> pair <speaker_mac_address>
> trust <speaker_mac_address>
> connect <speaker_mac_address>
> quit
```

After the previous steps the Bluetooth speaker should be connected as intended, and with the speaker used in thesis, the UE Roll 2, a sound confirmation can be heard when connection is successfully established. The Bluetooth speaker should automatically connect when powered on, if the Raspberry Pi is already active. Otherwise, if Raspberry Pi is booted and the speaker is already on, the speaker can be restarted as well. A script to automatically execute this can also be used, which runs the steps shown above.

7.8.2 Audio output

As the integration support of ALSA (Advanced Linux Sound Architecture) and Bluez was removed after Bluez 5.0, the Bluetooth audio output needs to be handled with ALSA's Bluetooth integration *bluealsa*. Normally the conventional configuration would be adequate but during the developing one major error, or rather a problem was noticed. The Bluetooth speaker handles one audio stream with ease but when another audio stream is directed to the same speaker, for example from TTS module, the audio vanishes from the latter source, as the device is already in use. As the sauna uses Snips AI and various TTS produced sounds and normal audio files, the problem of having only one audio stream at one time was unsuitable.

Simultaneous audio streams were handled with JACK Audio Connection Kit. JACK is a sound server daemon to provide sound integration across several platforms. In this case, JACK is used to handle the ALSA sound, combining different audio sources together, enabling to have Mopidy, TTS and other sounds working in one whole from one speaker.

JACK is installed as *apt*-package. This is the easiest way to install and the configuration is done afterwards.

```
# Install jackd
$ sudo apt-get install jackd
```

The configuration for the ALSA to use JACK as audio output is done in ALSA's configuration file in */etc/asound.conf*. (Saunaxio – ALSA 2018)

After configuration of the ALSA to use JACK by default, the JACK service must be started in a specific way to use ALSA as the audio backend. A script made for this purpose runs in the background and starts the *jackd* service whenever the Bluetooth speaker is active and available for use.

```
# Start jackd, executed in /etc/rc.local
...
su pi -c 'nohup bash /home/pi/scripts/audio/jacktivate.sh >
/dev/null 2>&1 &'
...
```

The script is executed as *pi* user, as other services using the audio output and JACK may fail if executed without the specified user. The service specified configuration considering the audio is described on the said services chapter.

Script used for JACK enables audio output only through Bluetooth device specified in */etc/asound.conf*. Should the user desire the audio output through audio cable from 3.5mm jack, the script needs to be modified and the *device* must be removed in order to activate the default of audio jack. The modified script does not update automatically; a reboot is needed after modifications.

```
# Edit /home/pi/scripts/audio/jacktivate.sh
...
jackd -t 10000 -p 500 -d alsa -r 32000 -n 3 -p 2048
...
```

The command has a timeout *-t* of 10 000 milliseconds and a maximum of *-p 500* ports available for JACK to manage. The audio interface backend *-d* is set to *alsa*,

which has additional options of sample rate *-r* of 32 kHz, number of periods of playback latency *-n* 3 and number of frames between JACK process calls 2048.

The Bluetooth device is normally specified after *alsa* as playback *"-P uerollraw"*, without this the default device, the audio jack, is activated as audio output. The output device for Bluetooth audio is specified in the */etc/asound.conf*.

7.8.3 Audio input

One of the main features is the speech recognition system used to control the Home Assistant components with voice while inside the sauna. The voice input requires the USB microphone correctly configured to function with the Raspberry Pi's audio setup and for the Snips to recognise the audio input device.

The USB microphone is connected normally to the Raspberry Pi, either directly or, in this case, with an extension cable to bring the microphone closer to the user. The microphone and speaker placement are done in a manner that speaker does not affect the recognition of words for Snips AI, nor does it unintentionally activate the Snips AI with a probable hotword.

Once the microphone is connected to the Raspberry Pi the USB devices are listed with the *usb-devices* command. Best way to determine the correct device is to run the command, disconnect the USB and run the command again, to see which of the devices is absent. This information is not necessarily needed for the configuration in this setup to work.

The microphone is configured in the */etc/asound.conf* file, same as the audio output. The audio input is set as a *capture* device and to play through JACK. Both JACK and Snips AI have *capture_ports* defined to make service hear the audio. The full configuration for the microphone in *asound.conf* is found in the project's Git repository. (Saunaxio – ALSA 2018)

7.8.4 Text-To-Speech

The TTS is mainly used by the Snips AI and the Home Assistant to notify the user and to provide answers to the users enquires. The TTS is provided by the Festival Speech Synthesis System, which is installed as *apt* package, together with other needed components.

```
$ sudo apt-get install festival festvox-suopuhe-common festvox-
suopuhe-lj festvox-suopuhe-mv
```

The Festival can be used with either echoing the text to Festival or writing the text into Festival's CLI. The latter method is not used in thesis, as the easier way is to *echo* the wanted text into Festival.

The configuration file for the Festival is in */etc/festival.scm*. Festival uses *aplay* to play the sound and because of the JACK configuration will not work properly as default. *Aplay* is given parameters to properly play the Festivals files. This is done at the end of the configuration file. (Saunaxio – Festival 2018)

```
# Edit /etc/festival.scm
...
;;
(Parameter.set 'Audio_Command "truncate -s +8000 $FILE && cat
$FILE | aplay -D jack -q -c 1 -t raw -f s16 -r $SR")
(Parameter.set 'Audio_Method 'Audio_Command)
...
```

Aplay uses JACK as the device for audio output and *raw* as file type. The file format by default is *s16*. The sample rate is as variable, coming from the JACK in *jacktivate.sh* script described earlier. The *truncate* command is needed to increase the size of the audio file, if not applied the voice will cut out as the file ends too early.

By default, Festival will be speaking English. To test Festival is in working order the following can be typed, provided the sound configuration of before is done properly.

```
# Test Festival with echo
$ echo "Hello World" | festival --tts
```

Using Finnish language, or any other in that matter, arguments can be given to the Festival in the command above.

```
# Finnish language
$ echo "Hei Maaailma" | festival --language finnish --tts
$ echo "Jyväskylä" | festival --language finnish --tts
```

While this method works with words without the diaeresis letters (å, ä, ö), once these letters are applied the Finnish voice will leave those letters out, replacing them with unknown characters. This can be corrected with following method.

```
$ echo "Jyväskylä" | iconv -f UTF-8 -t ISO8859-1 | festival --
language finnish --tts
```

By converting the text from UTF-8 encoding to ISO8859-1 encoding, the diaeresis letters can be heard as intended. Because this method produces a long command to be used, in thesis work an own command was made for this Festival setup. The command *puhu*, Finnish for *speak*, is located in the */usr/bin/puhu* file with the following content. The name for the command should be something unique, as for example *speak* may be used by some other software and thus create complications between the two.

```
# New file /usr/bin/puhu
#!/bin/bash
sanat=$@
sudo -u pi bash -c "echo \"\$sanat\" | iconv -f UTF-8 -t
ISO8859-1 | festival --language finnish --tts"
```

This command allows the use of the Festival TTS by using the *puhu* in place of *echo*.

```
$ puhu Jyväskylä
```

The TTS is used in notifications for the users, from either Node-RED or Home Assistant. The TTS is set to speak only Finnish, which in cases of English

sentences will speak with a Finnish accent. This accent is hard to understand, due to which all notifications are set to Finnish.

While the speaker is active, e.g. Mopidy is playing music, the TTS produced messages can be heard but depending on the volume of the speaker itself, may be rather quiet. The problem is noticed, when TTS activates while the speaker is playing audio, and Festival's voice is barely heard over the audio. The Festival supports an in-built ability to increase the volume and it is defined in the Festival's configuration file. The volume can be changed accordingly to the speaker, in this case value of *0.9* was used.

```
# Edit /etc/festival.scm
...
;; Increase the volume for Festival
(set! default_after_synth_hooks
  (list
    (lambda (utt)
      (utt.wave.rescale utt 0.9 t))))
...
```

The Festival has a problem when activated numerous times in a short period of time. One way of recreating this is to activate and deactivate the microphone for the Snips AI by pushing the button repeatedly. This causes the TTS messages to jam the Festival and at the same time also the JACK audio server, as too many requests are being handled at the same time. This problem is avoided by turning the TTS messages that are used more often into audio files, in this case to *.wav* files. These audio files can be played with *aplay* and, with an addition to the *festival.scm* configuration file, are generated with the *puhu* command described earlier. The script used to generate *.wav* files is found in the Git repository. (Saunaxio – scripts 2018)

```
...
;; Generate audio files with 'puhu'
(Parameter.set 'Audio_Command "truncate -s +8000 $FILE && mv
$FILE /home/pi/Music/tts/audio")
...
```

The line above needs to be commented out when the normal *puhu* command is used. If not, *puhu* will instead only write audio files and no sound can be heard.

7.9 Mopidy

Mopidy is the music player used in the solution. Mopidy can be used to play music from either local device, a stream, such as YouTube, or from Spotify. The latter method is the most used and preferred one.

Mopidy is installed as *apt* package, along with the used extensions. The configuration for the Mopidy is described later.

```
$ wget -q -O - https://apt.mopidy.com/mopidy.gpg | sudo apt-key
add -
$ sudo wget -q -O /etc/apt/sources.list.d/mopidy.list
https://apt.mopidy.com/stretch.list
$ sudo apt-get install mopidy
$ sudo systemctl enable mopidy.service
```

By default, the Mopidy service is running under the *mopidy* user but because of the configuration done for the ALSA and JACK, *mopidy* is not allowed to access the audio output. Therefore, the service must be started as *pi* user, this can be changed in */lib/systemd/system/mopidy.service* by changing the *User* under [Service] to *pi*.

Mopidy supports many extensions that can be installed similarly to the main software or with *pip*. The extensions vary from different file and mixer extensions to backend and frontend extensions. The YouTube, Spotify and MusicBox Webclient were installed as backend and web extensions. The installed web extension is used to access the Mopidy service from a browser, in this case *sauna.local:6680/musicbox_webclient*.

```
# Show available extensions
$ pip search mopidy
$ sudo pip install Mopidy-YouTube Mopidy-Spotify Mopidy-
MusicBox-Webclient Mopidy-Moped Mopidy-Mopify
```

The YouTube extension allows streaming from YouTube content to Mopidy by providing the URL address. The Spotify requires a non-Facebook linked user

credentials, as well as a *client_id* and *client_secret* provided on the Mopidy's webpage (Mopidy – Spotify authentication 2018).

The configuration file for the Mopidy is in */etc/mopidy/mopidy.conf*. The default configuration is generated at installation but must be modified for Mopidy to function as intended. The essential changes to the configuration are following, with correct audio output, Spotify and access configurations. The full configuration file is available on the Git repository. (Saunaxio – Mopidy 2018)

```
...
[audio]
output = jackaudiosink

[mpd]
enabled = true
hostname = ::
port = 6600
max_connections = 200
connection_timeout = 600000

[http]
enabled = true
hostname = ::
port: 6680

[local]
enabled = true
media_dir = /own/music/directory/here
excluded_file_extensions = # enable all of the default

[youtube]
enabled = true

[spotify]
enabled = true
username = spotify_user_name
password = spotify_password
client_id = client_id
client_secret = client_secret
private_session = true
allow_cache = false
...
```

The MPD and HTTP are by default set to only listen to localhost, partly as a safety measure since no password is set and unwanted third-party members could gain access to the service. While only listening to localhost, the web client or the MPD

cannot be accessed. Therefore, the *hostname* is set to listen to all interfaces, the IPv4 and IPv6, which allows the web client to be accessed and used.

While the Mopidy-Musicbox-Webclient is the basic web client used, other web extensions were also installed, mainly to address the need for easier usage with, for example Spotify, or on mobile devices. One of such extensions was the *Mopidy-Moped*, a friendlier UI for mobile usage (Figure 50). This web client could easily be used to browse local audio files or locally saved streams from, e.g. YouTube. For easier Spotify usage the *Mopidy-Mopify* offered a Spotify-like UI, with the possibility to search own playlists or other playlists from Spotify, along with artist, albums and tracks (Figure 51). The latter also installs the *Mopidy-Local-Images*, which is used to present the pictures for album covers or artists in Spotify.

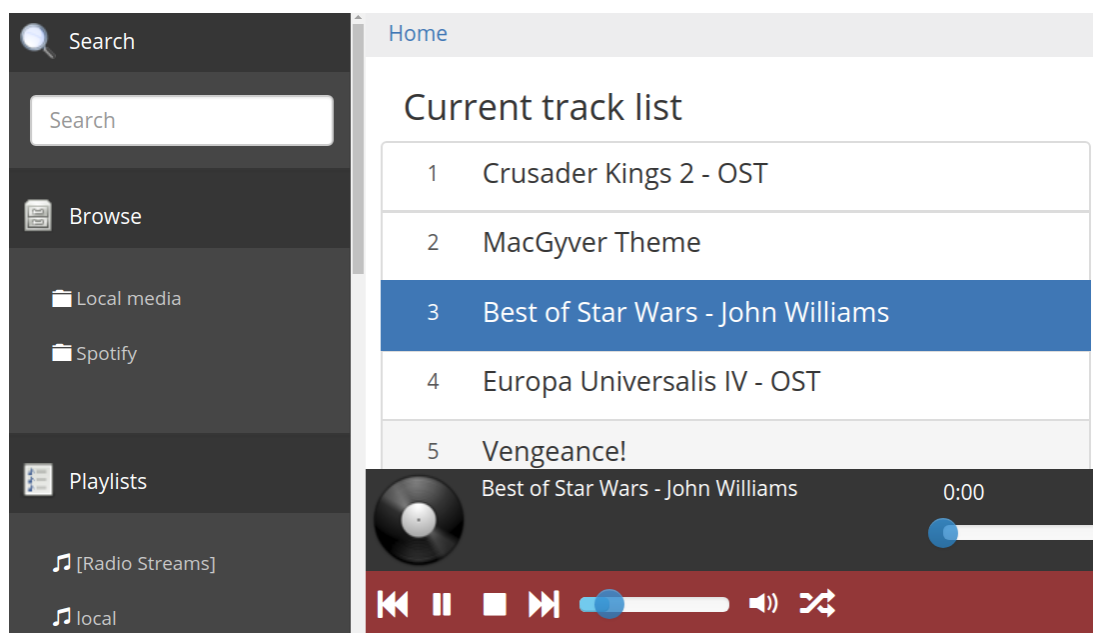


Figure 50. Mopidy-Moped UI

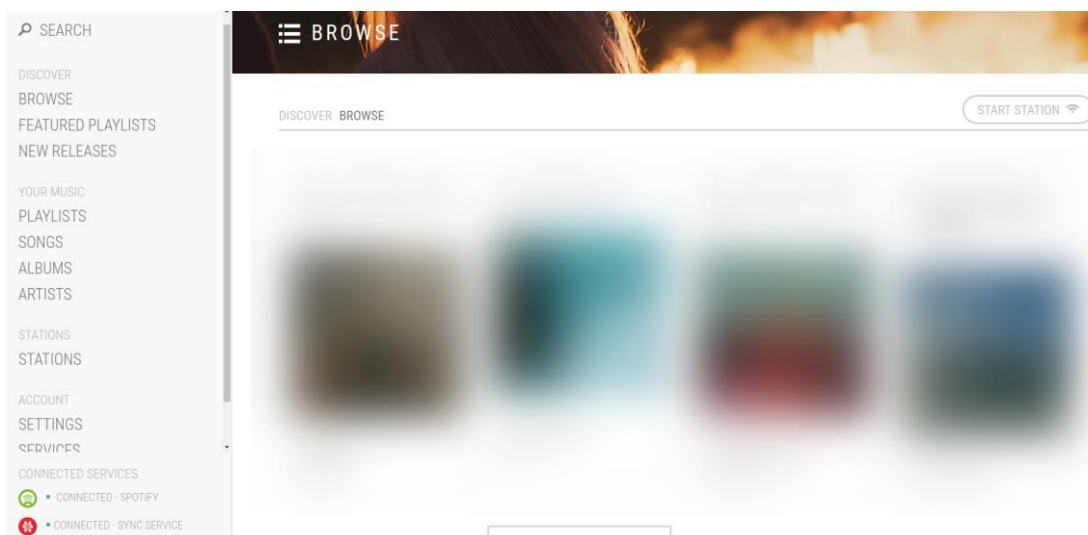


Figure 51. Mopidy-Mopify UI view

Adding local audio files proved to be complex procedure, as Mopidy must have certain privileges over the directory used for local files. For Mopidy to recognise the new files, Mopidy's *mopidyctl* can be used to scan for local files in the specified *media_dir* directory. A script was constructed, which could be run after adding new local audio files. Script is also available in Git repository. (Saunaxio – scripts 2018)

```
#!/bin/bash
chmod -R 777 /music/directory/here
sudo mopidyctl local scan
sudo chmod -R 777 /var/lib/mopidy
sudo service mopidy restart
```

The script, or the commands above, must be run whenever new addition are added to the *media_dir*, as the privileges of the directory change back to default after any modifications made.

While the Mopidy can be accessed and easily used from a web browser, the *Mopidy Mobile* application, available from e.g. Google Play, can be used for better mobile experience and easier use for the bather. The *Mopidy Mobile* scans for open MPD servers from local network and automatically connects to one. The application can be used in similar way to the web client on a browser. The application is also used to select the playlist when entering the sauna, as the speech recognition for a playlist is not supported in the setup.

During testing various audio files, *.wav*, *.opus* and *.mp3*, were used along with *Midi* files. Playing other audio files than the *.mid* ones was easy, and no problems occurred during the play. When playing the *.mid* files, the Raspberry Pi's sound cut off, due to hard load on the processor and soundcard. This was noticed while all other services were running on the Raspberry Pi and may result of high CPU and memory usage on the device.

7.10 Snips AI

The installation of the Snips AI was done several times due to problems with installation guides and device related matters. The installation method used installs the Snips AI platform with a few modifications to the original installation guide.

```
$ sudo apt-get install dirmngr -y
$ sudo bash -c 'echo "deb
https://raspbian.snips.ai/$(lsb_release -cs) stable main" >
/etc/apt/sources.list.d/snips.list'
# Find key D4F50CDCA10A2849 from pgp.mit.edu
$ sudo apt-key add
# Add key, Ctrl+D to save
$ sudo apt-get install snips-platform-voice -y
```

Installation can also be done with Snips' CLI manager Sam (Snips Assistant Manager). The Sam is installed with *npm* on external machine in the same network with the Raspberry Pi. It can be used to install the Snips AI platform, Snips skills and the Snips Assistant. This method is not described in detail, as it was not used in setup.

7.10.1 Configuration

After the installation the configuration must be correctly executed in order to activate the Snips' services. By default, the configuration can be found in */etc/snips.toml*, where every service can be configured separately. The main concern in the configuration file is to have the Assistant and MQTT broker

configured properly. The full configuration file is found on the project's Git repository. (Saunaxio – Snips AI 2018)

The MQTT broker is installed separately, as described before, and all services using the MQTT must use the same broker running at *1883* port. Snips AI is configured to use this broker in the configuration file.

```
[snips-common]
bus = "mqtt"
mqtt = "localhost:1883"
assistant = "/usr/share/snips/assistant"
```

The *assistant* is the Snips Assistant created in the Snips Console and the path, where the Assistant is installed, by default under */usr/share/snips/* directory, as seen above. This can be changed to any directory, provided the Assistant is located in the accordingly.

The *hotword* for Snips AI is set in the Snips Console from a few version, the one used in the setup being "*Jarvis*", for this was the most easily recognised by the microphone for Snips, and easy to pronounce with Finnish accent. During the writing Snips added the possibility of implementing own hotwords, however, this was not done for the setup.

```
[snips-hotword]
model = "/usr/share/snips/assistant/custom_hotword"
sensitivity = "0.9"
```

Hotword is located under Assistant, in this case the *custom_hotword* is used instead of the default one. The *sensitivity* is upped to *0.9* from default of *0.5* to make Snips' hotword detection more sensitive in recognising hotwords. With the hotword of "*Jarvis*", the Snips AI will also react to very similar words, such as "*Varis*" or "*Jari*", due to higher sensitivity.

Snips AI has in-built TTS service, which can be used to notify the user about various matters and to ask additional questions or confirmations for executable intents set in the Snips Console. The TTS is selected from three options, with the *customtts* being the one used in this case. The *puhu* command created for the

Festival's TTS is also used by Snips AI to allow possible communication by Snips' TTS service.

```
[snips-tts]
provider = "customtts"
customtts = { command = ["puhu", "%%TEXT%%"] }
```

While not often used as notification channel, the TTS service is still configured to be available for use when needed. Most of the notifications to the user are done in the Node-RED and Home Assistant as *shell_commands*.

The Assistant has in-built audio files for *start_of_input*, *end_of_input* and *error* notifications defined in */usr/share/snips/assistant/custom_dialogue/sound/* directory, which can be replaced with any other audio file, preferably of a *.wav* format. The audio files are configured under the same directory in *config.json*, should the name of the audio files change. Having exceedingly long audio files is not advised, since the *snips-audio-server* will prevent the other services from activating and the usage of Snips will suffer.

Audio notifications are disabled in Snips by default and must be activated via MQTT message to the *hermes/feedback/sound/toggleOn* topic, with the message of the current *siteId*, where Snips audio feedback is to be activated.

```
$ mosquitto_pub -h localhost -p 1883 -t
hermes/feedback/sound/toggleOn -m '{"siteId":"default"}'
```

The audio notifications can be deactivated by sending the message to *.../toggleOff* topic. Snips AI still stays active and functioning, only the audio notifications are not heard. This also affects the in-built TTS but as this is not used, replaced by Festival TTS' *puhu* via Home Assistant, it is not a main concern.

7.10.2 Assistant

The Snips AI's Assistant is created in the Snips Console from a browser environment. The Snips Console requires registration with a valid email address, as the verification is sent to said email. Once the profile is created user can create new Assistant, with a unique name, in case of creating several different assistants.

The language used is selected from the available options, in this case English was used.

After creation the usable *hotword* is selected, as described earlier. At this point new *skills* are added from existing bundles, or by creating a new skill from scratch (Figure 52).

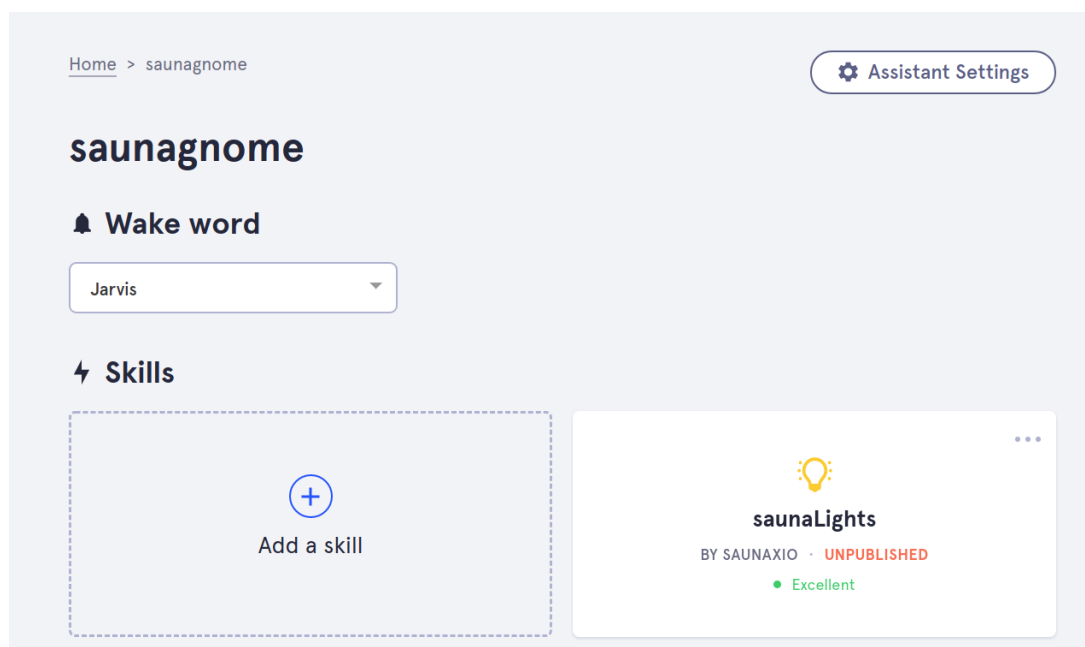


Figure 52. Creating skills in Snips Console

Each *skill* consists of *intents*, which in addition can hold several *intent slots* to more precisely handle the actions triggered by Snips. In this setup the *slots* were not used, though some of them were created for future possibilities.

The Assistant's skills are trained by *training examples* created by the user or by community for a fee. The training examples are written in the same language the Assistant is configured to use, as set before in Assistant creation. Official skill bundles found in the store can hold up to 1 000 training examples, the more examples the more precise and better the Assistant is with the said intent. The skills created for the setup hold around 20 training examples, which is enough, if the variety of the examples is high (Figure 53).

lightsOff

1 Slot

NAME	TYPE	SLOT REQUIRED?	+ Add Slot
lightLocation	snips/def... builtin	<input type="checkbox"/> Slot Required?	

24 Training Examples

Automatically generate additional training examples
Generate

Turn the lights off, please
+ Add Training Example

Figure 53. Training the skill with training examples

After saving the created intents, the Assistant is trained, which can be seen on the right side of the UI. After training has finished, the newly created Assistant can be tested by writing queries to the Assistant. The Assistant should response by showing the corresponding intent with the *probability* of how certain the Assistant is about the captured input (Figure 54). After a successful query, the Assistant is downloaded and installed either manually, the way described below, or with Snips Assistant Manage (Sam), which is not described in the thesis work.

 Turn the lights off, please

```
{
  "input": "turn the lights off please",
  "intent": {
    "intentName": "SaunaxIO:lightsOff",
    "probability": 1
  },
  "slots": []
}
```

Figure 54. Query for the new skill

The Assistant is downloaded as an *.zip* file, which, once copied onto Raspberry Pi with *scp*, can be unzipped with *unzip* command. To ease the burden of installing the Assistant in the correct directory, to */usr/share/snips/*, a script was made, to which the name of the Assistant could be given as an argument. The script can be found on the projects Git repository. (Saunaxio – Assistant install 2018)

```
$ sh scripts/snips/installSnipsAssistant.sh proj_xxxxxxx
```

The script removes the old Assistant directory from */usr/share/snips/*, unzips and installs the new Assistant to the same directory, restarting the services and activating the audio for Snips.

7.10.3 Using Snips AI

After the Assistant is installed, by default in the */usr/share/snips/*, Snips is ready for use with voice commands. The Assistant *skills* are used from *intent_scripts* in Home Assistant, which handles the actions inside the service (See Chapter 7.6.2).

Snips is activated with a *hotword*, in this case "*Jarvis*". After the hotword is recognised an audio confirmation is played, if sound feedback is active, and the voice command can be given to Snips. If the *intent* is recognised from the captured sentence, another audio confirmation of a successful recognition is

heard. If not, an error sound is played, after which the Snips returns back to hotword detection. A successfully captured *intent* will activate the corresponding actions from Home Assistant's *intent_scripts*. After a few seconds the Snips returns to hotword detection, waiting for a new command.

The Snips AI and its functions can be monitored with *snips-watch -vvv* command, where the *-vvv* define how much knowledge is shown to the user. This produces the output seen in the figure (Figure 55). If not installed by default, it can be installed as an *apt* package, similarly to other Snips services.

```
[16:39:14] [Hotword] detected on site default
[16:39:14] [Asr] was asked to stop listening on site default
[16:39:14] [Hotword] was asked to toggle itself 'off' on site default
[16:39:14] [Dialogue] session with id '4c6d8a2d-08ea-46e5-ac57-3a997c8ce6c3' was started on site default
[16:39:14] [AudioServer] was asked to play a wav of 84.2 kB with id '1e0938de-cc87-47aa-a3e2-33733c7767d5' on :
[16:39:15] [AudioServer] finished playing wav with id '1e0938de-cc87-47aa-a3e2-33733c7767d5'
[16:39:15] [Asr] was asked to listen on site default
[16:39:22] [Asr] captured text "what time is it" in 7.4s
[16:39:22] [Asr] was asked to stop listening on site default
[16:39:22] [AudioServer] was asked to play a wav of 93.1 kB with id '6e10f24a-a59e-493f-a93d-9fec502377d9' on :
[16:39:23] [AudioServer] finished playing wav with id '6e10f24a-a59e-493f-a93d-9fec502377d9'
[16:39:23] [Nlu] was asked to parse input what time is it
[16:39:23] [Nlu] detected intent SaunaxIO:TellTimeAndDate with probability 0.780 for input "what time is it"
[16:39:23] [Dialogue] New intent detected SaunaxIO:TellTimeAndDate with probability 0.780
[16:39:29] [Dialogue] session with id '4c6d8a2d-08ea-46e5-ac57-3a997c8ce6c3' was ended on site default.
[16:39:29] [Asr] was asked to stop listening on site default
[16:39:29] [Hotword] was asked to toggle itself 'on' on site default
```

Figure 55. Snips speech recognition flow

For a sequence diagram of Snips AI see Appendix 3.

7.10.4 Assistant installation problem

After an update to the Snips Console during April 2018, downloading the Assistant proved to be harder than before. The own, newly created skills were operating correctly, however, some of the intents in the bundles installed from the store had a problem and the download of the Assistant failed (Figure 56).

The error was tracked to the intents in the ready-made bundles by Snips, as creating own skill, with imported intents from the ready bundle, failed to download as well. This error happened with the *Music Player* by Snips, which was needed for the Home Assistant's MPD component to control the Mopidy music player. Creating own skill to control the MPD removed the error.

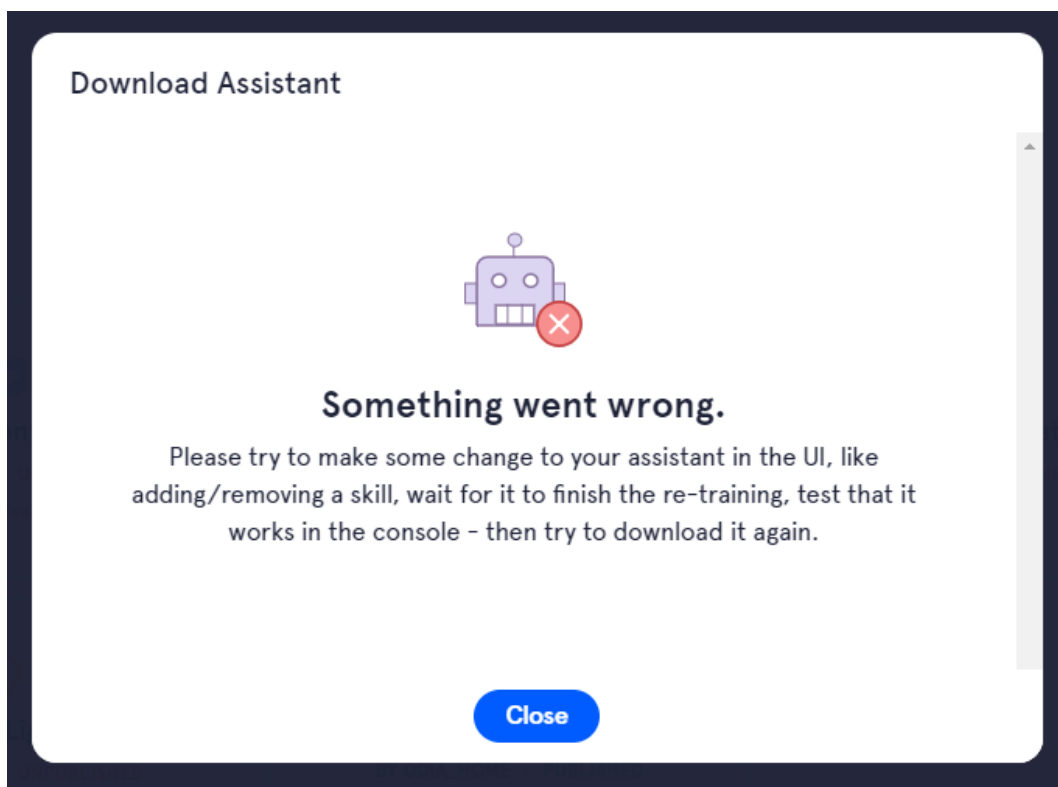


Figure 56. Message shown after download fails

While the error would occur from time to time, when new version of the Assistant was being downloaded, most of the problems could be sorted by removing all of the skill in the Assistant and adding them back one by one, as stated in the error message, while testing the download after every skill added. This method works only if the first added skill was added from a bundle rather than from own created skills. No notice of such problems was found on the Snips' community and the problems could be with either the Snips' servers or with user's own skills, however, when creating the skills, no shown errors occurred, which could imply this was the case.

Adding a skill for the MPD created by other user than Snips also worked as intended and was used in the setup for better recognition quality. The same occurred with the other skills, the Smart lights, Time and Date and Weather, as these were also added from store's bundle.

8 Evaluation

The project was finished on schedule during Spring 2018. At the end of the project, the system had been functioning for a month with the latest setup, with little to none maintenance periods. The stability of the system had thus been tested and the problems had been either directly addressed or noted for future development.

The final stage of the project was the installation of the system to the sauna environment, done outside the official project time, due to which no documentation of the actual installation was written. However, possible problematics noticed with the installation or with the system itself were to be noted with the client, later to be addressed by either the writer, client or potential future developer.

The initial functional requirements (See Table 1) for the system were fulfilled almost without any modifications.

The system was able to operate independently and steadfastly (RE100), with ability to address problems related to Node-RED's flow errors or loss of the Internet connection, as well as with ability for the user to restart services from the Home Assistant's UI. The bather was able to apply water on the stove either manually by pressing a button or a switch on the wall, or with a voice command to the Snips AI (RE200, RE210), which then executed wanted actions through Home Assistant's components. These components could control any device connected to the Home Assistant, whether physical or not. Control over the lights was possible with either Home Assistant's UI or with a voice command (RE400, RE410). Similarly, the system was able to answer to enquires made by the user, concerning the temperature, humidity or current time (RE300, RE301, RE302).

The user was able to control the music player with either *Mopidy Mobile* application for smartphone or with voice commands, controlling the volume, setting next or previous track and pausing and resuming the track (RE500, RE510, RE511, RE512). The possibility of selecting a playlist with just a voice

command was not accomplished, as the setup used with MPD and Home Assistant did not support this. As a workaround, the playlist was selected from the mobile application (RE501) when entering the sauna, with the rest control over the music player being done by voice commands.

Automation for applying water onto stove (RE201) was not accomplished and was moved to the future development.

9 Future development

The project leaves many possibilities for future development, acting as a basic platform to where add components, services and functionalities. The future development was discussed with the client during the project, as the ever growing pool of ideas were not executable in the period of time given for the project.

Such additions for the sauna were some sort of HUD or reflection screen to show data on, as the visible screens or any other technology inside the sauna was not desirable. A better sound system for the sauna was also on the list, together with more ambient environment with help of more clever usage of the Snips AI and Festival's TTS service.

The integrity of the existing system could be improved, creating the possibility to use Snips AI through *snips-skill-server* rather than through Home Assistant's *intent_scripts*. This could improve the amount of available controllable objects, as well as make voice controlling easier to do, with no more need for the Home Assistant's involvement.

The client had also a vision of a sauna conference implementation. In this vision the bather could take part in conference meeting with other people, who would also be seated in a sauna environment at the time.

These and many more improvements, additions or modifications would be done by future bachelor's thesis writers or other project workers. The base platform

created in the project offers good standings for future development, with either the same tools or new that may be better suited for the platform at given time.

10 Conclusion

Building an IoT solution for a sauna environment proofed to be an interesting assignment. The 'normal' IoT projects made, by others and by writer himself, were all designed for 'normal' surroundings. These projects were usually accessible at any given time, should something fail in the system, being as simple as losing the Internet connection. The device could be accessed and the problem could be addressed. With the device located in a sauna, the system had to be different.

The sauna provided extra problems when considering the usability and functionality in sauna's environment. As the device had to be independent, even for long periods of time, the approach in developing such system was different from normal. This created some headache for the writer when trying to map out the needed services to achieve such goal. Luckily, the writer was not alone with the matter, as the client provided great ideas over the period of developing the solution.

The challenges provided by the project, some even provided by the writer himself, helped to achieve a better understanding of IoT systems, whether they be located in a sauna or in normal room environment. The challenges faced during the development provided fine knowledge for a young Software Engineer starting his career in the big world. The previously acquired knowledge from studies proofed to be helpful, especially the knowledge gained from different projects during the past four years and from past two summers in Challenge Factory and WIMMA Lab.

As a thesis work, the project was a great introduction to different technologies and tools available for use when considering creating an IoT solution of any kind. It even created an interest of building some sort of automation and remote-control system to writer's own home, as the use of Home Assistant proofed to be

easier than it at first seemed. The independent work at the beginning of the project created fine knowledge and increased the interest regarding the assignment. Though not everything ended up as intended at the first place, the final solution was still something to be proud of.

References

- Barron, A. 2014. Pizza as a Service. 30.7.2014. Accessed 3.4.2018. Retrieved from <https://www.linkedin.com/pulse/20140730172610-9679881-pizza-as-a-service/> .
- Benchoff, B. 2016. Introducing the Raspberry Pi 3. 28.2.2016. Accessed 25.2.2018. Retrieved from <https://hackaday.com/2016/02/28/introducing-the-raspberry-pi-3/> .
- Brown, E. 2016. Home Assistant: The Python Approach to Home Automation. 20.6.2016. Accessed 20.3.2018. Retrieved from <https://www.linux.com/news/home-assistant-python-approach-home-automation-video> .
- Custom UI – Home Assistant. 2018. Accessed 9.4.2018. Retrieved from <https://github.com/andrey-git/home-assistant-custom-ui> .
- DigitalOcean – Pricing. 2018. Accessed on 2.4.2018. Retrieved from <https://www.digitalocean.com/pricing/> .
- DigitalOcean. 2018. Accessed on 15.2.2018. Retrieved from <https://www.digitalocean.com> .
- Edmonds, M. Chandler, N. n.d. How Smart Homes Work. Accessed 24.3.2018. Retrieved from <https://home.howstuffworks.com/smart-home1.htm> .
- Festival – Festvox. 2017. Accessed 20.3.2018. Retrieved from <https://github.com/festvox/festvox> .
- Gartner Newsroom. 2017. Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017. 7.2.2017. Accessed 5.3.2018. Retrieved from <https://www.gartner.com/newsroom/id/3598917> .
- Gite, V. 2012. Difference of Authentication and Authorisation. 28.2.2012. Accessed 2.2.2018. Retrieved from <https://www.cyberciti.biz/faq/authentication-vs-authorization/> .
- Google – App passwords. 2018. Accessed 20.2.2018. Retrieved from <https://support.google.com/accounts/answer/185833> .
- Google 2-Step verification. 2018. Accessed 20.2.2018. Retrieved from <https://support.google.com/accounts/answer/1085463> .
- Grafana alert notifications. 2017. Accessed 4.3.2018. Retrieved from <http://docs.grafana.org/alerting/notifications/> .
- Grafana Labs. 2018. Accessed 25.2.2018. Retrieved from <https://grafana.com/grafana> .

Groover, M. P. 2018. Automation. 16.3.2018. Accessed 26.3.2018. Retrieved from <https://www.britannica.com/technology/automation> .

Hefnawi, A. 2014. 3.5.2014. MAX6675 driver and code. Accessed 3.4.2018. Retrieved from <https://github.com/draco003/max31855> .

Home Assistant – Components. 2018. Accessed 20.3.2018. Retrieved from <https://www.home-assistant.io/components/> .

InfluxDB. 2018. Accessed 24.2.2018. Retrieved from <https://www.influxdata.com/time-series-platform/influxdb/> .

Install Home Assistant. n.d. Accessed 14.2.2018. Retrieved from <https://www.home-assistant.io/getting-started/> .

Internet Sauna & Steam Room Alert. 2016. 19.7.2016. Accessed 19.11.2017. Retrieved from <https://www.scandiamfg.com/internet-sauna-steam-room-alarm/> .

IoT History. 2016. 1.2.2016. Accessed 13.9.2017. Retrieved from <https://www.postscapes.com/internet-of-things-history> .

Jousimaa, O. 2017. Node-RED RuuviTag node. 4.7.2017. Accessed 22.2.2018. Retrieved from <https://github.com/ojousima/node-red> .

Lämpötila ja löyly. 1997. Accessed 14.2.2018. Retrieved from <http://www.saunaside.com/tekstit/loyly.htm> .

Light, R. n.d. MQTT man page. Accessed 22.3.2018. Retrieved from <https://mosquitto.org/man/mqtt-7.html> .

Loponen, T. 2005. Suopuhe. 11.11.2005. Accessed 20.3.2018. Retrieved from <https://www.tivi.fi/Arkisto/2005-11-11/Suopuhe-3082175.html> .

Mäkinen, J. 2016. Having Fun With IoT. 12.11.2016. Accessed 25.11.2017. Retrieved from <http://www.juhonkoti.net/2016/11/12/having-fun-with-iot> .

Marjamaa, M. 2016. Flow: More with less. 17.10.2016. Accessed 25.11.2017. Retrieved from <http://flow-morewithless.blogspot.fi/2016/10/iot-for-saunas-zwave-thermometer.html> .

Material Design Icons. n.d. Accessed 23.3.2018. Retrieved from <https://materialdesignicons.com/> .

Miller, P. 2017. Raspberry Pi sales. 17.3.2017. Accessed 24.1.2018. Retrieved from <https://www.theverge.com/circuitbreaker/2017/3/17/14962170/raspberry-pi-sales-12-5-million-five-years-beats-commodore-64> .

Mopidy – Spotify authentication. 2018. Accessed 23.2.2018. Retrieved from <https://www.mopidy.com/authenticate/#spotify> .

Mopidy. 2018. Accessed 15.2.2018. Retrieved from <https://www.mopidy.com> .

Noble – BLE central module. 2015. Accessed 3.3.2018. Retrieved from <https://github.com/noble/noble#running-without-rootsudo> .

Node-RED. 2017. Accessed 20.3.2018. Retrieved from <https://nodered.org/> .

Official RuuviTag Firmware. 2017. Accessed 23.2.2018. Retrieved from <https://lab.ruuvi.com/ruuvitag-fw/> .

OpenWeatherMap – API. n.d. Accessed 4.4.2018. Retrieved from <https://openweathermap.org/api> .

PVC Heat Distortion Temperature. n.d. Accessed 13.2.2018. Retrieved from <http://www.pvc.org/en/p/heat-distortion-temperature-softening-temperature> .

Raspberry Pi 3 Pinout. 2016. 2.6.2016. Accessed 4.3.2018. Retrieved from <https://myelectronicslab.com/raspberry-pi-3-gpio-model-b-block-pinout/> .

Rouse, M. 2016. Internet of Things. July 2016. Accessed 25.10.2017. Retrieved from <http://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT> .

Ruuvi: Weather Station. 2016/2017. Accessed 20.2.2018. Retrieved from <https://f.ruuvi.com/t/official-ruuvi-app-weather-station/17/8> .

RuuviTag. n.d. Accessed 19.2.2018. Retrieved from <https://tag.ruuvi.com/> .

Saunan kehitystä. 1997/1999. Accessed 16.3.2018. Retrieved from <http://www.saunasite.com/index-fi/kehitys.htm> .

Saunan rakentaminen – Poistoilma. 1997/1999. Accessed 16.3.2018. Retrieved from <http://www.saunasite.com/index-fi/poisilma.htm> .

Saunan rakentaminen – Tuloilma. 1997/1999. Accessed 16.3.2018. Retrieved from <http://www.saunasite.com/index-fi/tuloilma.htm> .

Saunaxio – ALSA. 2018. Accessed 21.5.2018. Retrieved from <https://github.com/saunaxio/Saunaxio/tree/master/audio/alsa> .

Saunaxio – Assistant install. 2018. Accessed 21.5.2018. Retrieved from <https://github.com/saunaxio/Saunaxio/tree/master/scripts/snips> .

Saunaxio – Festival. 2018. Accessed 21.5.2018. Retrieved from <https://github.com/saunaxio/Saunaxio/tree/master/audio/festival> .

Saunaxio – Home Assistant. 2018. Accessed 21.5.2018. Retrieved from <https://github.com/saunaxio/Saunaxio/tree/master/home-assistant> .

Saunaxio – Mopidy. 2018. Accessed 21.5.2018. Retrieved from <https://github.com/saunaxio/Saunaxio/tree/master/audio/mopidy> .

Saunaxio – NodeRED flow. 2018. Accessed 21.5.2018. Retrieved from <https://github.com/saunaxio/Saunaxio/tree/master/node-red> .

Saunaxio – scripts. 2018. Accessed 21.5.2018. Retrieved from <https://github.com/saunaxio/Saunaxio/tree/master/scripts/audio> .

Saunaxio – Snips AI. 2018. Accessed 21.5.2018. Retrieved from <https://github.com/saunaxio/Saunaxio/tree/master/snips-ai> .

Saunaxio – thermocouple. 2018. Accessed 19.5.2018. Retrieved from <https://github.com/saunaxio/Saunaxio/tree/master/codes/thermocouple>

Saunaxio – ultrasonic. 2018. Accessed 19.5.2018. Retrieved from <https://github.com/saunaxio/Saunaxio/tree/master/codes/ultrasonic> .

Smoke sauna. 2004. 28.3.2004. Accessed 16.5.2018. Retrieved from https://upload.wikimedia.org/wikipedia/commons/b/b8/Smoke_sauna.JPG .

Snips Voice Platform. 2018. 14.2.2018. Accessed 22.3.2018. Retrieved from <https://github.com/snipsco/snips-platform-documentation/wiki> .

Tuohy, J. 2015. What is home automation and how do I get started?. 26.1.2015. Accessed 10.10.2017. Retrieved from <https://www.networkworld.com/article/2874914/internet-of-things/what-is-home-automation-and-how-do-i-get-started.html> .

What is cloud computing. n.d. Accessed 25.10.2017. Retrieved from <https://www.interoute.com/what-cloud-computing> .

What Is Home Automation and How Does it Work?. N.d. Accessed 9.10.2017. Retrieved from <https://www.safewise.com/home-security-faq/how-does-home-automation-work> .

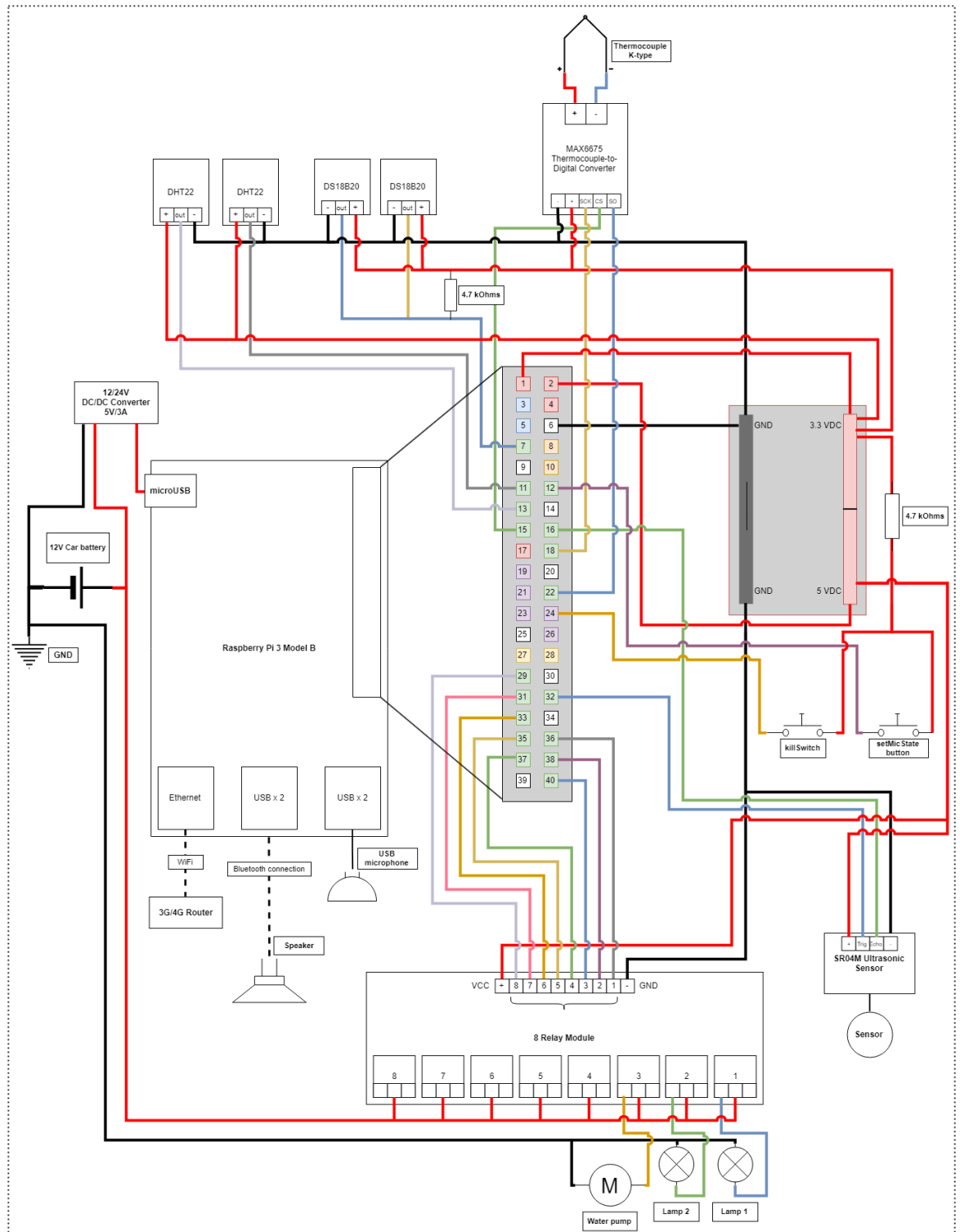
What is IaaS. n.d. Accessed 25.10.2017. Retrieved from <https://www.interoute.com/what-iaas> .

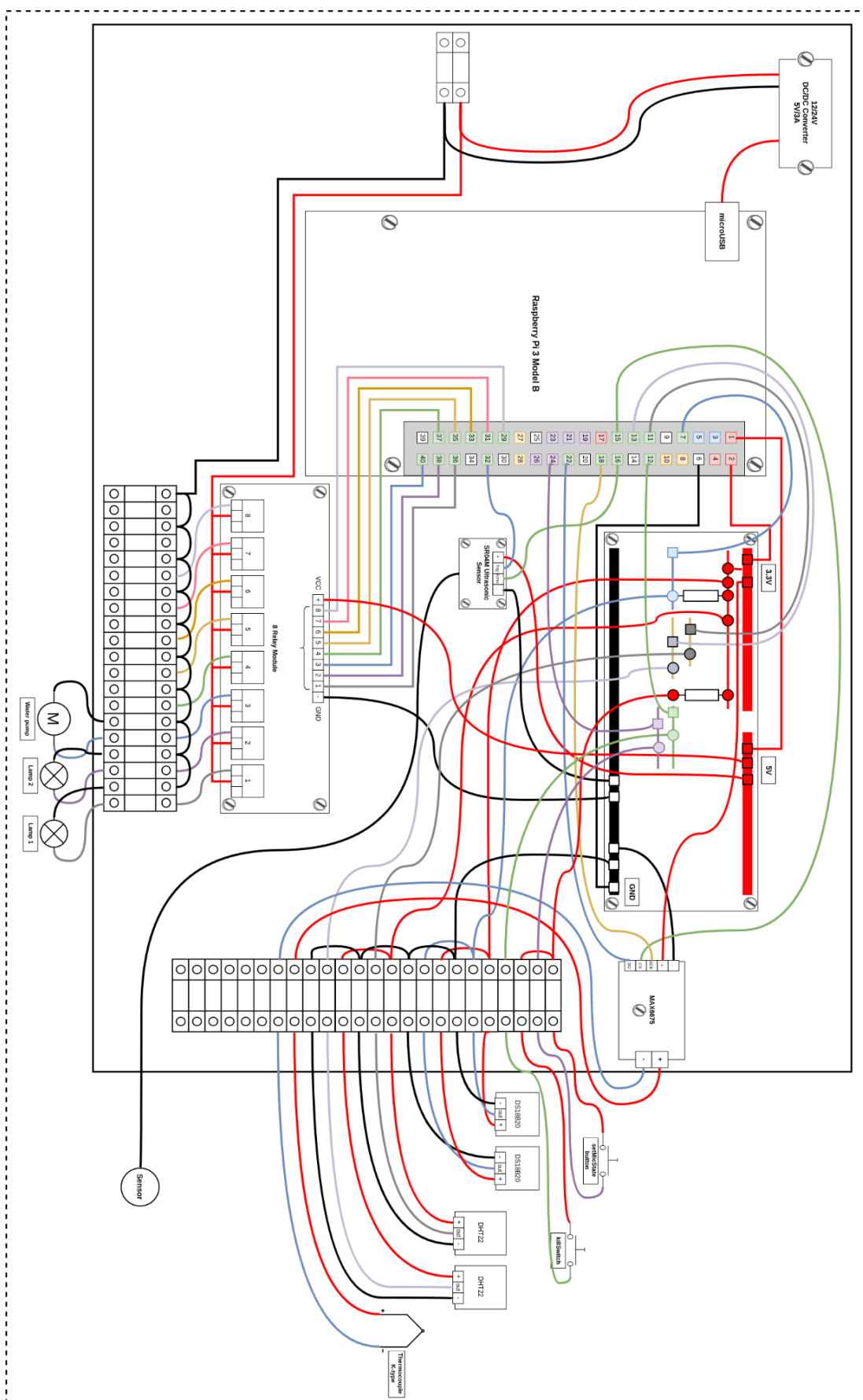
What is PaaS. n.d. Accessed 25.10.2017. Retrieved from <https://www.interoute.com/what-paas> .

What is SaaS. n.d. Accessed 25.10.2017. Retrieved from <https://www.interoute.com/what-saas> .

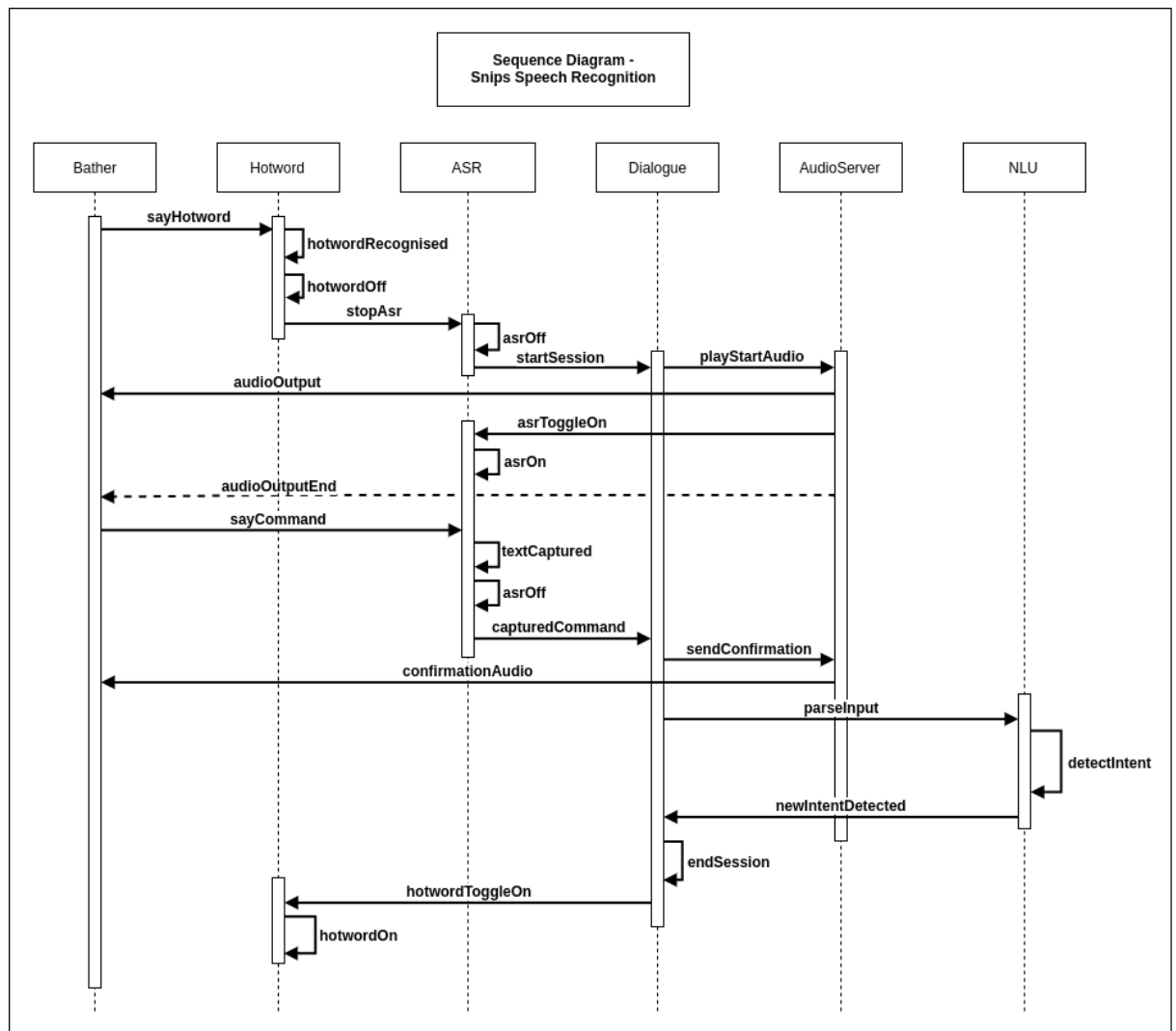
Appendices

Appendix 1. Block diagram of the system





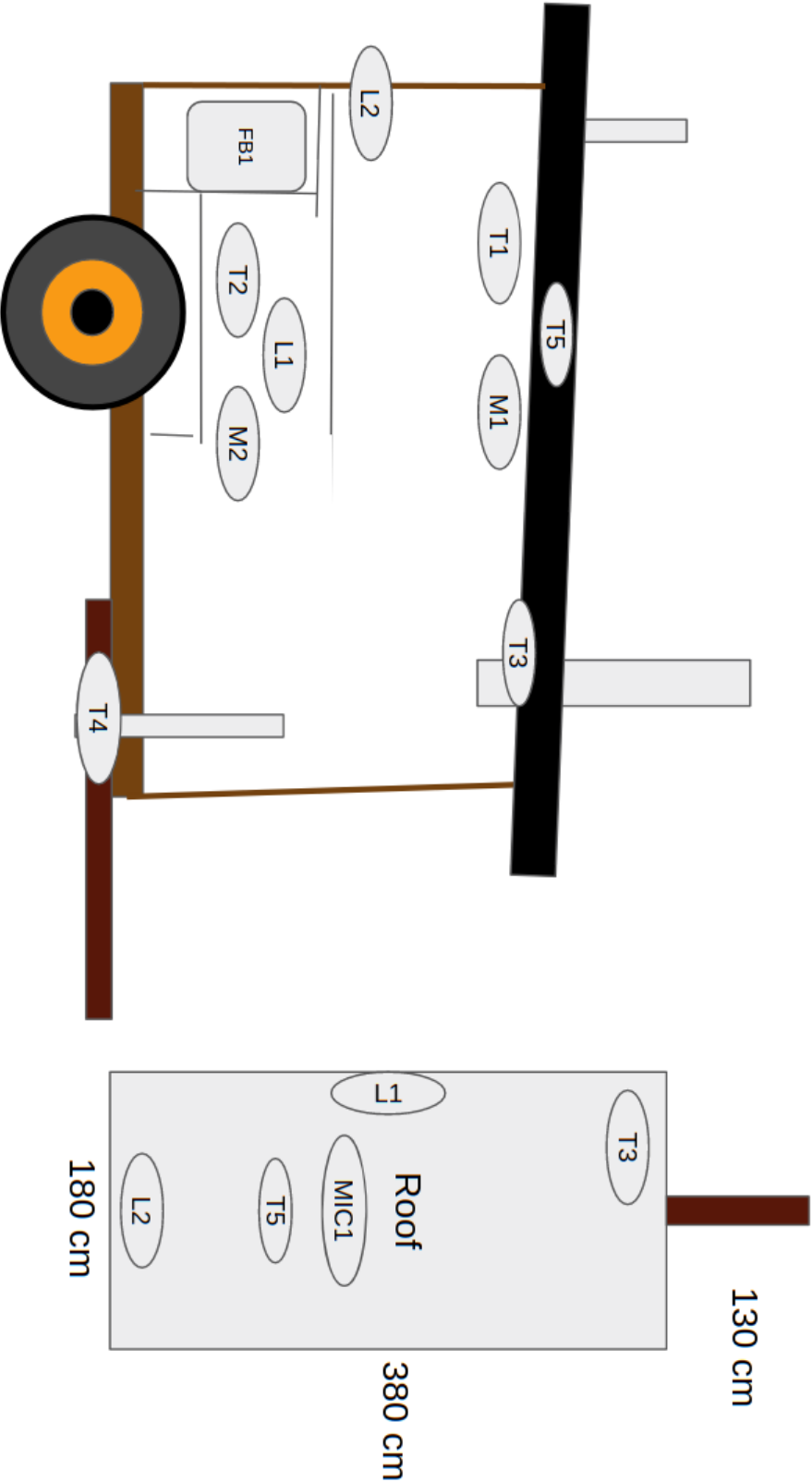
Appendix 3. Sequence diagram of Snips AI function



Appendix 4. Hardware list

ID	Function	Description	Location
M1	Moisture	DHT22	Upper left wall
M2	Moisture	DHT22	Lower left wall
T1	Temperature	DS18B20	Left wall
T2	Temperature	DS18B20	Concrete, below stove
T3	Temperature	Thermocouple K-type	Roof, above stove
T4	Env. sensor	RuuviTag	Air income pipe
T5	Env. sensor	RuuviTag	Roof insulation
SP1	Speaker	UE Roll 2	Under sauna benches
MIC1	Microphone	USB microphone	Close to bather
US1	Ultrasonic	AJ-SR04M	Top of water tank
RL1	Relay	8-channel board	Fieldbox
CON1	Converter	Step-Down converter	Fieldbox
BT1	Button	Microphone activation	Close to bather
BT2	Button	Emergency kill switch	Wall
TB1	Term. block	Terminal block	Fieldbox
CB1	Circuit board	Circuit board	Fieldbox
FB1	Fieldbox	Fieldbox	Sauna back end
WP1	Water pump	Water pump	Near water tank
L1	Light	Light 1	Left wall
L2	Light	Light 2	Back wall

Appendix 5. Hardware placement inside the sauna



Appendix 6. QR-code for the Github repository (URL included)



Saunaxio – Github repository <https://github.com/saunaxio/Saunaxio> .