Gintaras Pacevičius

# Smart house
## Engineering an IoT system

Bachelor's thesis
Bachelor of Engineering

2019


South-Eastern Finland
University of Applied Sciences

| Author | Degree | Time |
|---|---|---|
| Gintaras Pacevičius | Bachelor of Engineering | May 2019 |
| **Thesis title**<br><br>Smart house – Engineering an IoT system | | 42 pages<br>15 pages of appendices |
| **Commissioned by** | | |
| **Supervisor**<br><br>Reijo Vuohelainen | | |


**Abstract**

Smart houses introduced people with home automation that resulted in many positive features for home users. Using Internet of Things devices and computer networks, home automation and remote access became core features of a smart house. The objective of this thesis was to build and study a custom built smart house solution.

To complete the objectives a practical implementation method was used. This method allowed testing out how smart house market-related problems were addressed in a custom built smart house solution. Smart house market-related problems that were studied during the thesis project implementation were scalability, flexibility, security, price and the complexity of the solution.

This thesis project implementation was completed successfully. A custom built smart house solution was created and studied. The study showed that created solution was scalable, flexible, secure, cheap, but complex to build for users that lacks knowledge about the information technology and smart house field. The more scalable, flexible, secure the smart house solution was designed to be, the more complex it became to implement.



**Keywords**

Smart house, Internet of Things, Raspberry Pi 3

**TABLE OF CONTENT**

# 1. INTRODUCTION

A smart house is a home that has automated systems to control and monitor any function of a house, for example, lighting, temperature, security, air quality control, etc. Smart house technology allows users to control their connected devices using the applications from smartphones, or using the web browsers on other network connected devices. The main purpose of this control is to enhance comfort levels for home users.

According to the article by Rouse M. (2019) smart house technology originated in 1975 with the release of X10, a communication protocol for home automation. The implementation of X10 used electric wiring to programmable outlets or switches. Electric signals conveyed commands to devices, controlling how and when they operate. However, as electrical wiring is not designed to be particularly free from radio-band “noises”, X10 was not always fully reliable, meaning that some signals would be lost. Another problem for X10 was that it was one-way technology, meaning that systems could take commands, but could not send data back to the central network.

Modern smart house technology uses Internet of things (IoT) devices to create systems that automates specific home functions. These systems communicate with the local server using network media. The local server acts as the central control point. Modern technology offers more types of media for devices’ communications. It also offers two-way communications, meaning that devices not only receive commands, but also send data back to the local network. (Rouse M. 2019.)

According to the article by Francoise C. (2018) smart houses struggle to become mainstream because of its technical nature. (Francoise C. 2018). It is due to the fact that smart houses have price, flexibility, security, complexity problems. There are smart house solutions that can be bought or rented, for example, GoogleHome, ZigBee, Jung, Eclipse, eNet, etc. Companies like to make their solutions as user-friendly as possible by simplifying systems and offering live technical support. Some companies require making contracts for technical support. Then in cases of failure, companies claim the responsibility for fixing the problems. This results users being charged, even if the problem

was simple to fix, and users could do it by themselves. Everything adds up, resulting an expensive solution to maintain. This is an example of the price problem that smart houses have.

Due to the simplification process performed by the companies, a portion of configurations are also denied or removed from the users. This can lead to problems when users cannot achieve specific goals. These goals could include scaling an infrastructure, having different or more automation solutions, designing a custom security, etc. This is one example of the flexibility or scalability problem that smart houses have.

Some popular vendors require internet access to operate the smart house solution. This creates a vulnerability, because quite a lot of these solutions offer physical security controls features. If users can control their security digitally and remotely, so can an unauthorized intruder. Users are afraid of more skilled people in the internet who can take over their systems. This is an example of the security problem that smart houses have.

To avoid problems that companies' solutions result in, and to build a smart house solution on their own, users must deal with a more complex approach. Software and hardware related issues can be difficult to resolve for average users. The whole system depends on the users' abilities to keep it running. There is a lot of documentation about the smart house topic. However, frequent updates can make most of it outdated. This is an example of the complexity problem that smart houses have.

A custom smart house solution built by the users is more complex, but it also offers more solutions for solving scalability, flexibility, security and price problems. The thesis objective is to build and research a custom built smart house solution. The study needs to address the problems that are caused by smart houses offered in the marked. These problems include scalability, flexibility, security, price and complexity.

This thesis paper contains 5 chapters. Chapter 1 contains introduction that summarizes smart house technology and its current problems. Chapter 2 contains information related to a background study that summarizes the

information technology used during the thesis project implementation. Chapter 3 contains information related to the thesis project implementation that summarizes the steps of how custom smart house solutions was built. Chapter 4 contains results that summarizes how smart house problems mentioned were addressed during the thesis project implementation. Chapter 5 concludes the thesis paper.

# 2. BACKGROUND STUDY

This chapter deals with smart houses, computer networking and single board computer technologies. Computer network technology includes sub-sections about computer devices, local area networks, wireless networks, network protocols, MQTT protocol and virtual private networks. The section about single board computers introduces Internet of Things devices and printed circuit boards. The purpose of the background study is to display the technology that is used throughout the thesis project.

## 2.1. Smart house technology

According to the article by Yi Man Li et al. (2016) Various methods have been proposed for the advancement and development of sophisticated control systems of smart house Environment. Smart air-conditioners, security devices, mobile phones and home theatres put theoretical smart house into practice. Some of them utilize the technology of artificial intelligence (AI), multi-agent systems and automation control. These smart house features improve users' quality of life. In addition, some smart house systems can help disabled people to have a better and more convenient life. There are three generations of smart houses.

The first-generation of smart houses are using devices to monitor users' activities and operate electrical devices in a predefined pattern. These systems mainly use Bluetooth automation solutions which connect electrical devices to the Bluetooth controller. The system lowers the needs of electrical wires and intrusive electrical installation via wireless media. However, sharing one single Bluetooth between large number of devices leads to delay issues. Possible solutions for the delay issues are, for example, ZigBee-based

automation network. To operate it requires a local proxy server. It enables communication between electrical appliances and the local area network. Another possible way to solve the delay issues may involve the Internet of Things approach which uses an all-IP solution based on IPv4 or IPv6 protocols.

The second generation of smart houses evolved into Smart House Environments (SHE). In addition to the devices that are used in first generation smart houses SHE exhibits various forms of AI by improving traditional home automation systems. It increases comfortableness, lowers operation costs and enhances security. With the help from automation, AI and multi-agent systems there are a lot of methods for designing SHE for the users.

The third generation of smart houses involves robots. Affordable robots are equipped with artificial intelligence and can respond to users' needs via voice recognition. For example, Zenbo connects to smart house devices, moves freely and independently around the house, sees things via its camera, makes video calls, recognizes faces, takes photos and videos. Robots as such not only control smart things but can also be viewed as a friend "who" can interact with the user. (Yi Man Li et al. 2016.)

## 2.2. Computer networking

According to the article by Cisco Computer Networking Academy (2019, Overview to network components) a computer network is a digital telecommunication network. It allows nodes to share resources between each other. These nodes exchange data using data links (connections). These connections are formed using cable media such as wires or optic cables. Or, these connections are formed using wireless media such as radio communications.

All devices in the network are identified as nodes. This includes computers, smart phones, printers, routers, etc. Nodes are identified by network addresses. Each address is unique for each node. Computers in the network take the role of clients or servers, while networking devices take the role of

intermediary network devices. Servers are computers with software that enable them to provide information, like email or web pages, to other end devices on the network. Clients are computers with software installed that enable them to request and display the information obtained from the server. Switches, hubs and routers are mentioned intermediary network devices. They can retransmit data signals generated by nodes to other nodes in the network. By connecting multiple nodes together, a local area network (LAN) is formed. By connecting multiple LANs, a wide area network (WAN) is formed.

Computer networks support a huge number of applications and services. These include digital video, digital audio, instant messaging applications, electronic mails, storage devices, etc. Communication protocols are used to carry data signals in the network media. They organize network traffic, size and topology. The best-known computer network is the Internet. (Cisco Computer Networking Academy 2019.)

### 2.2.1. Computers devices

According to the publication of the Florida Center for Instructional Technology (2013) computers connected to a network are broadly categorized as servers or workstations. Servers are generally not used by humans directly, but rather run continuously to provide "services" to the other computers (and their human users) on the network. Services provided can include printing and faxing, software hosting, file storage and sharing, messaging, data storage and retrieval, complete access control (security) for the network's resources, and many others.

Clients are called such because they typically do have a human user which interacts with the network through them. Clients were traditionally considered a desktop, consisting of a computer, keyboard, display, and mouse, or a laptop, with integrated keyboard, display, and touchpad. With the advent of the tablet computer, and the touch screen devices such as iPad and iPhone, our definition of client is quickly evolving to include those devices, because of their ability to interact with the network and utilize network services.

Servers tend to be more powerful than clients, although configurations are according to requirements. A group of servers might be in a secure area, away from the users. In such cases, it is common for the servers to operate without a display or keyboard and be managed remotely through the network. Every computer on a network should be appropriately configured for its use. (Florida Center for Instructional Technology 2013.)

For example, one server computer is set up to share a service within the network with high availability. Number of clients are able to use this service any time without effecting any other clients within the network. This provides administrators the ability to keep server devices isolated in remote locations. These locations could be designed to have cooling, physical security, and other features that could be required.

### 2.2.2. Local Area Network

According to the article by Mitchell B. (2019) a local area network (LAN) is a network that is confined to a relatively small geographic area such as a laboratory, school, building or group of buildings. LANs are built to enable sharing of resources and services. A local area network may contain from one to many thousands of nodes. Some nodes like the servers stay permanently associated with the LAN while smartphones, laptop computers may join and leave any time.

There are plenty of advantages to LANs. The most obivous one is that software, files, or any other information can be shared with all the nodes that are connected within the LAN. This arrangement not only makes administration easier but it also reduces the cost of having to buy additional equipment. Since sharing is a major role of a local area network, it is clear that this type of network means faster communication. Files and other data can be shared much more quickly if they are within the internal network instead of the Internet. Sharing resources on a network requires central administrative control, which means it's easier to make changes, monitor, update, troubleshoot, and maintain those resources. (Mitchell B. 2019.)

For example, a local are network. Within this LAN there is a single server that is communicating with a home automation devices. The server acts as central management node for these devices. The LAN itself is isolated from the Internet and other users' specified LANs, which creates an additional layer of security since home automation traffic is separated from users' generated traffic. Topology like can be a core structure for smart house solution's network.

### 2.2.3. Wireless network

According to the article by Marshall B. et al. (2019) Many people use wireless networking, also called Wi-Fi or 802.11 networking, to connect their computers at home, and some cities are trying to use the technology to provide free or low-cost Internet access to residents. In the near future, wireless networking may become so widespread that users could access the Internet just about anywhere at any time, without using wires. Wi-Fi has a lot of advantages. Wireless networks are easy to set up and inexpensive.

A wireless network uses radio waves, just like cell phones, televisions and radios do. In fact, communication across a wireless network is a lot like two-way radio communication. A computer's wireless adapter translates data into a radio signal and transmits it using an antenna. A wireless router then receives the signal and decodes it. Finally the router sends the information to the Internet using a physical, wired Ethernet connection. The process also works in reverse, with the router receiving information from the Internet, translating it into a radio signal and sending it to the computer's wireless adapter.

Wireless LANs (WLANs) are convenient. They allow users to easily connect multiple devices and to move them from place to place without disconnecting and reconnecting wires. Most devices are with built-in wireless transmitters. If device is not equipped with wireless technology, users are able to use wireless adapters, microcontroller or other wireless technology to provide functionality. Once a wireless adapter and the drivers are installed device is able to discover existing WLANs. (Marshall B. et al. 2019.)

For example, a wireless microchip with full microcontroller capabilities is installed into printed circuit board. The device becomes Wi-Fi compactible, that allows it to communicate with the network using WLANs. Knowing that Wi-Fi does not require physical wires to operate, these devices can be installed into physical locations where wires would be an issue or would not be supported at all. This is the perfect technology for home automation devices in the smart house.

### 2.2.4. Network communication protocols

According to the presentation by Cisco Computer Networking Academy (Communication Fundamentals 2019) networks communicate like people do. There are three main requirements that must be met for successful communication. These are source, destination and media. In the network a signal, that originates at source node, is sent through network media, that can be wired or wireless, to the destination node. Data is not sent as plain text. It is transmitted into binary code by the source. Then this code is sent and decoded back into data by the destination when received.

Protocols are vital for effective communication. They define a common format and set of rules for exchanging messages between the nodes. They do so by defining sources, destinations, common language, grammar, speed, time of delivery, error checking and acknowledgements. Main parameters for a network protocol are message size, encoding, delivery options, timing fields, formatting and encapsulation. A set of protocols can that work together to provide comprehensive network communication services can form a protocol suite. Protocol suites may be specified by a standard, organization, or developed by a vendor. For example, The TCP/IP protocol suite is an open standard. The protocols are freely available. These protocols can be implement by any vendor on their hardware or in their software.

TCP/IP communication process starts when data is sent from a node to node. This starts the encapsulation with application layer protocol that sends the data to the transport layer. The transport layer breaks the data into segments and identifies each of them. Then the internet layer adds source and destination address information, creating a packet. Lastly network access

layer adds its Ethernet information forming an Ethernet frame, or data link frame. This frame is delivered to the nearest intermediary network device along the path towards the destination node. Each intermediary network device adds a new data link information before forwarding the packet to the destination. When receiving the data link frames, node processes and removes each protocol header in the opposite order it was added. The process starts by removing network access layer's Ethernet information, then internet layer's packet information, then transport layer's segment information, finally to process the data for the user by the application layer. (Cisco Computer Networking Academy - Communication Fundamentals 2019.)

### 2.2.5. MQTT protocol

According to the article by SmartHomeBlog (2018) MQTT is a protocol created by IBM that stands for Message Queue Telemetry Transport. MQTT is primarily focused on Machine-to-Machine (M2M) communications. Although MQTT is used for a number of applications, it is heavily used as a communication protocol for Internet of Things devices. The main reason for it is MQTT's design. MQTT is designed for applications where the required bandwidth is very low. It consumes very little resources, and it is available on many different platforms.

MQTT uses a Star Topology with a central node called Broker, and clients connected to it. The Broker is what runs the communications. It is in charge of sending and receiving the messages from the clients. The communication in the MQTT protocol is based on topics. A client can publish messages on a topic and all subscribers of that topic will receive that message.

MQTT can be implemented as communications protocol in smart house network. In such case, there would be two main elements. These elements would be MQTT broker (server) and clients (home automation devices). MQTT Broker allows the clients to publish messages to topics, or to use messages from topics. Clients can be sensors or actuators, where sensors publish messages on topics so that subscribers could read them, and actuators are subscribed to topics waiting for commands to execute an action. (SmartHomeBlog 2018.)

### 2.2.6. Virtual Private Network

According to the article by Hoffman C. (2019) organizations use virtual private networks (VPNs) to create an end-to-end private network connection over third-party network, such as the Internet. VPNs use a tunnel to enable remote users to access central site network resources. For VPNs to guarantee that the information remains secure while traversing the tunnel, modern cryptographic methods are applied. This allows to establish secure, end-to-end, private network connections. (Hoffman C. 2019.)

According to the article by Cisco Computer Networking Academy (2019) VPN is a communications environment in which access is strictly controlled to permit peer connections within a defined community of interest. Confidentiality is achieved by encrypting the traffic within the VPN. A secure implementation of VPN with encryption is what is generally equated with the concept of virtual private networking. (Cisco Networking Academy Introducing VPNs 2019.)

For example, a VPN service is running on the smart house network router. This VPN service allows securely access the local area network and its resources using Internet connection, that is public. Device, that is at remote location and forms a VPN tunnel, is considered as if it was connected directly to the LAN. This technology can be used to securely access smart house network using the Internet without creating vulnerabilities for the smart house network.

## 2.3. Single board computers

According to the article by Technopedia (2019) a single board computer (SBC) is a complete computer in which a single circuit board comprises memory, input and output, a microprocessor and other features. Unlike a personal computer, SBCs do not rely on expansions for other functions. SBCs reduces the system's overall cost as the number of circuit boards, connectors and driver circuits are all reduced. SBCs are designed differently from standard desktop or laptop computers, as they are completely self-contained. They often make use of a wide range of microprocessors and have increased density for the integrated circuits used.

Because they are designed to control simple processes, SBCs are slower and more limited compared to personal computers. However, there are many advantages to use SBCs. Their features are well integrated due to nearly everything being native to the machine. They are lighter in weight, compact in size, more reliable and much more power efficient then multi-board computers. SBCs are mostly used in embedded applications. They are also used in applications for process control, like complex robotic systems and processor-intensive applications. (Technopedia 2019.)

### 2.3.1. Internet of Things

According to the document by International Telecommunication Union (2012) internet of things (IoT) is a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies. Regarding the IoT, things are objects of the physical world (physical things) or of the information world (virtual things). Things are capable of being identified and integrated into communication networks. They have associated information, which can be static and dynamic.

Physical things exist in the physical world and are capable of being sensed, actuated and connected. Examples of physical things include the surrounding environment, industrial robots, goods and electrical equipment. Virtual things exist in the information world and are capable of being stored, processed and accessed. Examples of virtual things include multimedia content and application software. Through the exploitation of identification, data capture, processing and communication capabilities, the IoT makes full use of things to offer services to all kinds of applications, while still ensuring that security and privacy requirements are fulfilled. (International Telecommunication Union 2012.)

For example, an IoT physical thing could communicate using network media with a server. This physical thing could use MQTT topics to send or receive specific data of specific things (doors, windows, power plugs, etc.) that could be used for home automation. An IoT virtual thing, coded by the software of the home automation server, could interpret the data sent by the IoT physical

thing and perform automated action specified by users when certain conditions happen. It could send broadcast messages, sound the alarms, turn on or off some devices, and so on. This makes IoT one of the core elements of smart house technology.

### 2.3.2. Printed circuit boards

According to the article by Electical4U (2019) the printed circuit board (PCB) also called a printed wiring board is an insulating board as a base material, cut to some dimension with a conductive pattern and hole (such as component holes) to replace the pad of electronic devices, and make an interconnection between electronic components. PCB is an important electronic component – it is a supporter of electronic components and electrical connections. A PCB is mainly made up of a pad, mounting holes, wirings, components, connectors, fillings, electric boundaries and so on. With structure like that, PCBs have special advantages like high density, high reliability, designability, manufacturability, testability, mountablity, maintainability. These special advantages are the reasons why PCBs are very popular. (Electical4U 2019.)

For example, custom PCBs with specific components for wireless communication could be made to represent an IoT physical thing. The functionality of these IoT physical things could be designed using different additional components such as LED light strings, magnetic read sensors, etc. Designing IoT physical things that way, users are able to create numerous different IoT devices with different functionalities to create home automation for their smart houses.

## 3. PROJECT IMPLEMENTATION

This chapter includes a project implementation of a custom smart house solution. It contains details about software requirements, hardware requirements, network configuration, local server configuration, IoT device configurations, and whole project’s economical overview. Software requirements contains sub-sections “Open HAB 2”, “Home Assistant”, “Software for the project”. Hardware requirements contains sub-sections “Hardware for networking”, “Hardware for a local server”, “Hardware for IoT

devices”. IoT device configurations contains sub-sections “Door Sensor”, “LED light”, “Fire Sensor”.

## 3.1. Software requirements

To design a custom smart house solution, software as environment is required. It needs to support home automation and MQTT protocol broker. However, other optional requirements, such as available amount of documentation and active community, are also considered. For a practical implementation two popular environments are considered. The “OpenHAB 2”, and the “Home Assistant”. Practically, JAVA-based and Python-based solutions.

### 3.1.1. OpenHAB 2

According to the OpenHAB (2019) the Open Home Automation Bus is an open source, technology agnostic home automation platform which runs as the center of a smart house. OpenHAB is developed in Java and mainly based on the Eclipse SmartHome framework. It uses Apache Karaf together with Eclipse Equinox to create an Open Services Gateway initiative (OSGi) runtime environment. Jetty is used as an HTTP server. OpenHAB is a modular software that can be extended through add-ons. OpenHAB 2 is the latest version of the environment. Add-ons give OpenHAB a wide array of capabilities, from user interfaces to the ability to interact with a large and growing number of IoT physical things. Add-ons come from OpenHAB 1, OpenHAB 2 distribution, and the Eclipse SmartHome project. OpenHAB runs on many platforms, including Linux, Windows and Mac OSx.

The simplest way to use OpenHAB is to install OpenHABian operating system. OpenHABian offers a simple way to get up and running quickly, or complex approach for advanced users. The OpenHABian installation takes from 20 to 40 minutes. After that, it is possible to start configurations for the system using the WEB interface, or using a secure shell connection. Main configurations are performed using files inside the system. File configurations are written in using the Xbase syntax. IoT virtual things are coded by users

into the system. That allows users to add any device to their system. (OpenHAB 2019.)

According to the article by Brice A. (2018) OpenHAB has an active community and a lot of documentation. In case of problems, community is eager to respond and assist. There are many good practices which are documented, tested, and constantly updated. Updates for OpenHABian are rare, because of strict testing that is done, only polished updates are released. Rare update cycles make documentation last longer up to date. Rare update cycles also make the system more reliable and stable. (Brice A. 2018.)

### 3.1.2. Home Assistant

According to the Home Assistant Dev Docs (2019) Home Assistant is a Python program. It can be run on any operating system and it provides the ability to track, control and automate devices. Home Assistant is developed using Python 3 for the backend and Polymer (Web components) for the frontend components. As an open-source product, it is licensed under Apache 2.0.

HassBian is a combination of Home Assistant and tools that allows users to run it without setting up an operating system first. HassBian is an all-in one-solution and has a management user interface that can be used from the Home Assistant frontend. The installation process takes about 10 minutes. However, after the installation is completed, HassBian uses internet connection to update the system. Configuration bases on the click and install method. The system uses automatic scans for connected devices to add them into the system for home automation. It is an extremely user-friendly approach and supports devices from the majority of vendors. However, there is no easy way to add non-supported device to the system. If users want to add a custom device to the system, it can take up to ten lines of code for few different files to describe the device and its functionality in the system. File configurations are made using YAML syntax. YAML is a human-readable programming language. For Python users, AppDaemon software can be used to code non-supported devices using Python syntax, instead of using YAML. (Home Assistant Dev Docs 2019.)

According to the article by Brice A. (2018) Home Assistant gained popularity because of its simple click to install method for supported devices. However, it does not have as much documentation up to date and a community that is as active as OpenHAB's. This is because Home Assistant is relatively new compared to OpenHAB. Another reason for documentation issues is Home Assistant's update cycles. It happens every two weeks. Updates add support for new devices and fixe bugs. However, sometimes it introduces more bugs. These bugs can frequently make the system unstable or even broken. For example, if users add non-supported devices to the system and at some point an update that adds official support goes live the whole system becomes unstable. This may result in automatic scans adding duplicate devices to the system or worse. (Brice A. 2018.)

### 3.1.3. Software for the project

With possible options know, a comparison is created. The purpose of this comparison is to select the software that is going to be the core software for home automation server in the smart house solution. Full OpenHAB and Home Assistant comparison is in Table 1.

Table 1. OpenHAB and Home Assistant comparison

| **Function** | **OpenHAB** | **Home Assistant** |
|---|---|---|
| **Installation** | Guided installation<br>Takes 20 to 40 minutes<br>No need for internet access | Guided installation<br>Takes up to 10 minutes<br>Requires internet access |
| **Flexibility** | Users are able to add any devices to the system using add-ons or coding | Users are able to add devices to the system that are vendor supported using automatic scans, or any other non-supported devices using code. |
| **Community** | Huge | Growing |
| **Updates** | Stable updates only<br>Rare update cycles | Frequent updates include support for new devices faster |
| **Automation** | Java script or Xbase syntax | YAML or Python syntax |

Home assistant is a good solution. It has super user-friendly UI and constant support for devices. Developers keep on adding new devices to the system. The system is easy and quick to install. However, constant update cycles and lack of up to date documentation can lead to problems. OpenHAB is a better solution because of its stability. Since PCBs are used for the project as IoT physical things, they have to be coded into the system. Xbase requires less lines of code and less files to accomplish this compared to Home Assistant's YAML. Also OpenHABian installation progress is simple as well, it only takes more time. OpenHAB has more and better up to date documentation which makes this solution a perfect candidate for a custom smart house solution.

## 3.2. Hardware requirements

To design a custom smart house solution, hardware for networking, for a local server and for IoT devices is required. The main requirements for networking hardware is that Wi-Fi technology, the DHCP server, VPN server, firewall, and routing are supported. The main requirements for a local server are network interface for connection, storage for OpenHABian operating system, and low energy consumption. The main hardware requirements for IoT physical devices are Wi-Fi technology, PCB, enclosure and specific components for device functionality such as a magnetic read sensor, LED strip, etc.

### 3.2.1. Hardware for networking

For the networking device, most requirements are met on one device such as Cisco or MikroTik routers. However, these enterprise grade devices are too expensive for the home users to implement. For a project that requires small scale network, there is no need for expensive enterprise grade devices. For these reasons, products that are used at home environments are considered to be used for this smart house solution's network infrastructure. These products consist of Edimax, Netgear and TP-Link vendor devices. The extended list of requirements for the networking device includes the following:

- Support for IEEE 802.11ac/n/a 5GHz and IEEE 802.11b/g/n 2.4GHz wireless standards
- Wireless security of 64/128-bit WEP, WPA, WPA2 encryption

- Interfaces:
    - 2, or more 10/100/1000Mbps LAN interfaces
    - 1, or more 10/100/1000Mbps WAN interfaces
- Firewall Security
- Guest Network
- VPN server

Networking device for this solution is chosen from TP-LINK Archer C7, Netgear AC1750, Edimax AC750 devices. Table 2 introduces the specifications for each device listed.

Table 2. Specifications of the networking devices

| **Device** | **TP-LINK Archer C7** | **Netgear AC1750** | **Edimax AC750** |
|---|---|---|---|
| **Wireless standards supported** | IEEE 802.11ac/n/a 5GHz<br>IEEE 802.11b/g/n 2.4GHz | IEEE 802.11ac/n/a 5GHz<br>IEEE 802.11b/g/n 2.4GHz | IEEE 802.11ac/n/a 5GHz<br>IEEE 802.11b/g/n 2.4GHz |
| **Wireless security** | 64/128-bit WEP, WPA/WPA2, WPA-PSK/WPA2-PSK encryption | 64/128-bit WEP, WPA/WPA2, WPA-PSK/WPA2-PSK encryption | 64/128-bit WEP encryption, WPA and WPA2 security |
| **Interfaces** | 4 10/100/1000Mbps LAN interfaces<br>1 10/100/1000Mbps WAN interface<br>2 USB 2.0 ports | 4 10/100/1000Mbps LAN interfaces<br>1 10/100/1000Mbps WAN interface | 1 x RJ-45 10/100M WAN port<br>3 x RJ-45 10/100M LAN port |
| **Firewall** | DoS, SPI Firewall<br>IP Address Filter/MAC Address Filter/Domain Filter | Integrated SPI and NAT firewall protection | SPI anti-DoS firewall |
| **Guest Network** | 2.4GHz and 5GHz guest networks | 2.4GHz and 5GHz guest networks | 2.4GHz and 5GHz guest networks |
| **VPN server** | PPTP, L2TP, IPSec | PPTP, IPSec | VPN pass-through (IPSec/PPTP) |
| **Price EUR** | 79.99 | 89.99 | 39.99 |

All the devices have support for required wireless technology, wireless security, guest network, the VPN server. All three devices have at least basic firewall implementation. Only the interfaces and prices are different. TP-Link and Netgear routers offer high speed networking using gigabit ethernet interfaces in exchange for higher price. For the project's network, there is no need for gigabit ethernet provided bandwidth. With that in mind, the only specification left affecting the choice is the price. Edimax AC750 is the cheapest router in the list. It meets all the requirements in exchange for lower possible network speed. That is why Edimax AC750 is the perfect candidate to be chosen to be the networking device for a custom smart house solution.

### 3.2.2. Hardware for a local server

For the smart house solution's local server requirements are low. Interface for LAN and storage for OS are required. Since the requirements are low, a virtual server can be run insted of a physical device. It is important to note that this local server would have to run constantly without shutting down. Enterpises can afford physical servers like these, where this local server could be virtualized. Home users, however, may not own physical servers. Devices such as laptops or desktop computers that could virtualize such a server would have to run constantly. Running such a device constatnly would consume a lot of energy. For these reasons virtualization may not be the best option for home users. Since requirements for a local server are low, a single board computer can represent the local server. This type of computer is cheap and can run constantly with low energy consumption.

The device to represent the local server is chosen from Raspberry Pi 3, BeagleBone Black, Asus Tinker Board single board computers. Table 3 displays the specifications for each considered SBC.

Table 3. Specifications of consider single board computers

| **Device** | **BeagleBone Black** | **Raspberry Pi 3** | **ASUS Tinker Board** |
|---|---|---|---|
| **Processor** | AM335x 1 GHz ARM Cortex A8 | 1.2 GHz 64-bit ARMv8 | ARM Cortex-A17 1.8 GHz modelis |
| **Storage** | 4 GB integrated | Micro SD card | Micro SD card |
| **RAM** | 512 MB DDR3 | 1 GB DDR2 | 2 GB DDR3 |
| **Wi-Fi** | No Wi-Fi adapter | 802.11 n | 802.11 b/g/n |
| **Documentation** | Medium | Large | Small |
| **Physical ports** | Micro-USB, USB 2.0, 10/100 Ethernet, HDMI | Micro-USB, 4x USB 2.0, 10/100 Ethernet, HDMI, CSI camera port, DSI display port | Micro-USB, 2x USB 2.0, 10/100/1000 Ethernet, HDMI, CSI camera port, DSI display port |
| **Price EUR** | 57.10 | 34.75 | 57.70 |

All three devices have storage for the operating system (SD card or integrated), and LAN interface for network. BeagleBone Black and ASUS Tinker Boards are more expensive and more powerful. However, these computers lack documentation. Since requirements are low, a more powerful computer is not the important factor. This means that the price is the main factor that determines the device for the local server. This indicates that Raspberry Pi 3 is the best fit for this local server.

### 3.2.3. Hardware for IoT devices

Besides different requirements for different fucntionality, all Internet of Things physical devices for this custom smart house solution have common requirements. These common requirements are wireless adapter modules, printed circuit boards, and control enclosures. Because printed circuit boards are used as the core of the IoT physical devices, it does not have integrated networking interface. For each such IoT device an ESP8266 Wireless adapter module is required to be welded.

For this solution three IoT physical devices are considered to be created. One of these devices is Door sensor. The device-specific requirement for this IoT device is the magnetic read sensor. Another IoT device is LED Light. The specific requirement for this device is LED light strip. The third and last IoT device is Fire sensor. It does not require additional component, but it requires an additional device. This device is the fire detector device that has interconnect feature. With IoT physical devices covered, additional hardware that allows to flash these devices is required. This hardware is Arduino UNO computer.

### 3.3. Network configuration

The scale of the designed custom smart house solution network is a small office / home office network. The router WAN interface is configured to receive an IPv4 address dynamically. With this option selected an ISP provider assigns a public IPv4 address for the WAN connection providing internet access for local area network. It is important to note that without a public IPv4 address VPN cannot be created. For this project VPN is not configured because of network address translation (NAT). Local apartment's network configuration does not provide public IPv4 address for its hosts. Therefore, VPN is not configured.

LAN interfaces are configured to connect to the 192.168.1.0 network with 255.255.255.0 (/24) subnet mask. The router's address is set to be a static 192.168.1.254 IPv4 address – the last host address in the subnet. To assign addresses for network-connected devices dynamically, DHCP service in the router is configured. DHCP server options include the address lease time duration of 30 minutes, IPv4 address range of 192.168.1.1 to 192.168.1.220, and the address information of the default-gateway. A static DHCP lease is created for the local server's MAC addresses of the LAN adapter (to be 192.168.1.100). This lease is configured to be permanent.

For IoT physical devices to access LAN where the local server is located, WLAN is created. WLAN name (SSID) is configured to be Namai-WiFi. The best channel for WLAN to operate is selected using site survey utility. WLAN WPA Pre-Shared Key encryption is configured. WPA2 (AES) is chosen as the

WPA unicast cipher suite. With the Pre-Shared key format configured to be the passphrase, a strong password is created for authentication.

With the WLAN Namai-WiFi a guest WLAN WiFi-Namai is also created. This WLAN's clients are isolated from WLAN Namai-WiFi clients. This means that guest WLAN connected devices cannot communicate with Namai-WiFi clients. Guest WLAN WPA Pre-Shared Key encryption is also configured. WPA2 (AES) is also chosen to be the WPA unicast cipher's suite. With the Pre-Shared key format configured to be the passphrase, a different strong password is created for authentication.

Both WLANs are configured to run on 2.4 GHz radio range. The Namai-WiFi SSID is not broadcasted by using the hide SSID option. This means that the solution's WLAN, which has a local server within, is not shown in the WLAN list for client devices. By configuring these two WLANs like that, the network provides security by isolating the smart house solution devices. In case an attacker manages to crack guest WLAN password, the main network will remain isolated and secure from the attacker's reach. The router's firewall is configured to be enabled. Also, Anti-DoS features are enabled to provide protection against the Ping of Death, Pings from WAN, Port Scan and Sync Flood attacks.

VPN server is not configured for the project. If a public IPv4 address was assigned by the local apartment network, VPN would be configured. The VPN server would provide an additional layer of security for this solution's network. To connect to the VPN server, client devices would have been required to be configured with specific information. This information would include public a IPv4 server address, username, password, certificate, and the type of VPN. With these configurations the solution's network would be accessed and used securely from the internet, since VPN encrypts all the traffic and isolates devices.

### 3.4. Local server configurations

First, the OpenHABian operating system is flashed into the SD card of the Raspberry Pi 3 computer, which represents the local server. To flash the

OpenHABian to the SD card, the balenaEtcher software is downloaded and installed on computer device. An image of the OpenHABian operating system is downloaded from the OpenHAB's official WEB site. After roughly 5 minutes of flashing process, SD card is safely ejected and mounted into the local server. The local server is connected to the network using ethernet cable. After SD is mounted, and the local server is connected to the network, power cable is connected allowing the device to power on.

The OpenHABian starts by running script that performs initial configurations for operating system. To see the script in action, terminal emulation software PuTTY is used. After roughly 25 minutes first time setup is successfully finished message appears. This message indicates that operating system is ready to be used. With operating system ready to be used, a WEB site of the local host on server can be accessed using network connected devices. To access the local server's WEB interface, an IP address of device with port number "8080" is entered. For example, the local server is accessed using "192.168.1.4:8080" string in WEB browser. Site welcomes users to the OpenHAB 2 service initial setup. There users choose one of four available packages. Packages of the setup are Simple, Standard, Expert and Demo. Expert package is selected to be set up. After roughly 5 minutes welcome to OpenHAB 2 site is now displayed instead of service initial setup site.

With expert package set up successfully device initial configuration can be performed. Using terminal emulation software and SSH (22) port, connection to device terminal is established. Using openhabian-config command, system updates, upgrades are performed. Then hostname, system locale, system time zone and passwords are set. Finally, an additional component Mosquitto is installed. Device is then restarted to apply the system changes. With device restarted, FireMotD optional configuration is performed. FireMotD displays an overview of all used components of Linux-based operating system. This configuration is performed using terminal. After commands are executed device is restarted to apply the changes. List of commands used includes:

- cd /opt/FireMotD
- sudo ./FireMotD -I -v
- sudo ./FireMotD -TF /opt/FireMotD/themes/FireMotD-theme-Gray.json

- sudo ./FireMotD -T Gray

First of four listed commands navigates user to /opt/FireMotD directory. Second of four listed commands generates the counters for the components used of the system. Third of four listed commands generates the theme for overview to be displayed. Fourth of four listed commands sets the device to use the generated theme for overview.

To test earlier installed the additional component Mosquitto's functionality MQTT.fx software is used. First a profile is created to specify MQTT broker profile settings. Broker address specified is an IPv4 address of local server. Broker port is specified as 1883. With this profile completed, software can establish connection to the local server. To test functionality a subscription is made to device using # symbol. This means that all MQTT messages received by the server will also be displayed at connected MQTT.fx software. By using publish tab, test message is generated and sent to the local server. Because component is installed correctly, this test message is displayed in the subscribe tab as received message.

With the Mosquitto up and running additional configurations are performed using the WEB site of the local host. To allow legacy binding add-ons, include legacy 1.x bindings option is enabled and applied in system configuration tab. With this option MQTT Binding (1.x) is installed to the system. This specific binding is located in BINDINGS tab of add-ons. Using terminal emulation software to connect to the server one additional configuration is performed in the binding's configuration file. Inside the configuration file URL for MQTT broker is specified with line <broker>.url=tcp://192.168.1.100:1883 (the IP address specified is binded permanently to local server using MAC address of network adapter). List of commands used to edit the mqtt.cfg file:

- cd /etc/openhab2/services/
- sudo nano mqtt.cfg

First of two listed commands navigates the directory of mqtt.cfg file. Second of two listed commands edits the file using nano text editor.

To connect to the system remotely using the internet connection the OpenHAB Cloud Connector add-on is installed and configured using the web interface. With the add-on installed the server is able to establish connection with the cloud. To configure the cloud to have access to the local server, the myopenhab.org web site is used. There user account is created and connection to the local server is established. To establish connection OpenHAB settings are specified. These settings include local server UUID and Secret information. Both files that have the information are located in the local server. Information is viewed using terminal emulation software. With settings set, local server is restarted to apply the changes. When the server restarts, state of device in the cloud website is changed from offline to online. List of commands used on local server's terminal:

- sudo nano /var/lib/openhab2/uuid
- sudo nano var/lib/openhab2/openhabcloud/secret

First of two listed commands opens UUID string containing file using nano text editor. Second of two listed commands opens secret string containing file using nano text editor.

Finally, to prepare the local server to add IoT virtual devices four files are created. These files contain the information of things, items, sitemaps, and rules. Files are created inside the local server using terminal emulation software and nano text editor. Files are created and saved in /etc/openhab2/things, /items, /sitemaps, /rules directories accordingly. Files are named using sitemap name dot file name format. For the project files are named as home.things, home.items, home.sitemap, home.rules.

Things represent the physical layer of an OpenHAB system. From a configuration standpoint, Things tell OpenHAB which physical entities (devices, web services, information sources, etc.) are to be managed by the system. Physical entities are devices such as Z-Wave device. This file has little to know content due to lack of vendor supported devices in the project. Items represents the virtual layer for IoT devices. Items represent functionality that is used by the application (mainly user interfaces or automation logic). Items have a state and are used through events. The content of home.items

file includes Security System, Bedroom, LED Light and Home IoT virtual things. The code to define all these IoT virtual things is explained step by step at each of the IoT device's configuration sub-sections. Sitemaps are used to select and prepare elements such as Things and Items for various user interfaces. The content of home.sitemap file, represents frames for Security, LED Light, Fire Detector items. Security frame has two frames inside. Each of these frames contains Security System and Bedroom IoT virtual things. LED Light frame contains LED Light IoT virtual things. Fire Detector frame contains Home IoT virtual things. Rules are used for automating processes. Each rule can be triggered, which invokes a script that performs users' specified tasks. For example, turn on lights by modifying your items, send broadcast message to all users, etc. The code inside home.rules file is explained step by step at each IoT device configuration sub-chapter.

With virtual devices created and sitemap configured, user interface can be accessed using the WEB browser or android application. However, until IoT physical things are flashed and installed into their locations, no data is sent to the local server. Meaning that user interface can be viewed and accessed, but it does not provide any functionality yet. The content of home.things file is displayed on appendix 1. The content of home.items file is displayed on appendix 2. The content of home. sitemap file is displayed on appendix 3. The content of home. rules file is displayed on appendix 4. The content of user interface called Basic UI is displayed on appendix 5.

## 3.5. IoT devices configurations

For this custom smart house solution three IoT devices are configured using Arduino UNO device. IoT devices are connected to Arduino UNO single board computer using Male-to-Male cables. Arduino UNO single board computer is connected to workstation using USB type A cable. With devices connected with each other, using Arduino IDE software, it is possible to flash ESP8266 wireless modules that are welded on PCBs of IoT devices. Each IoT device is configured similarly using Male-to-Male cables, Arduino UNO device and Arduino IDE software.

### 3.5.1. Door Sensor

To configure Door sensor IoT device for this custom smart house solution device is prepared and binded with created IoT virtual things. First IoT physical device's configurations are performed. A jumper switch on PCB is manually switched from RUN to PGM mode. Then Door sensor IoT physical device is connected to Arduino UNO device using Male-to-Male cables. Then Arduino UNO Power section is cabled from RES to GND using Male-to-Male cable. Figure 1 displays Door Sensor IoT physical device's breadboard scheme connected to Arduino UNO device.

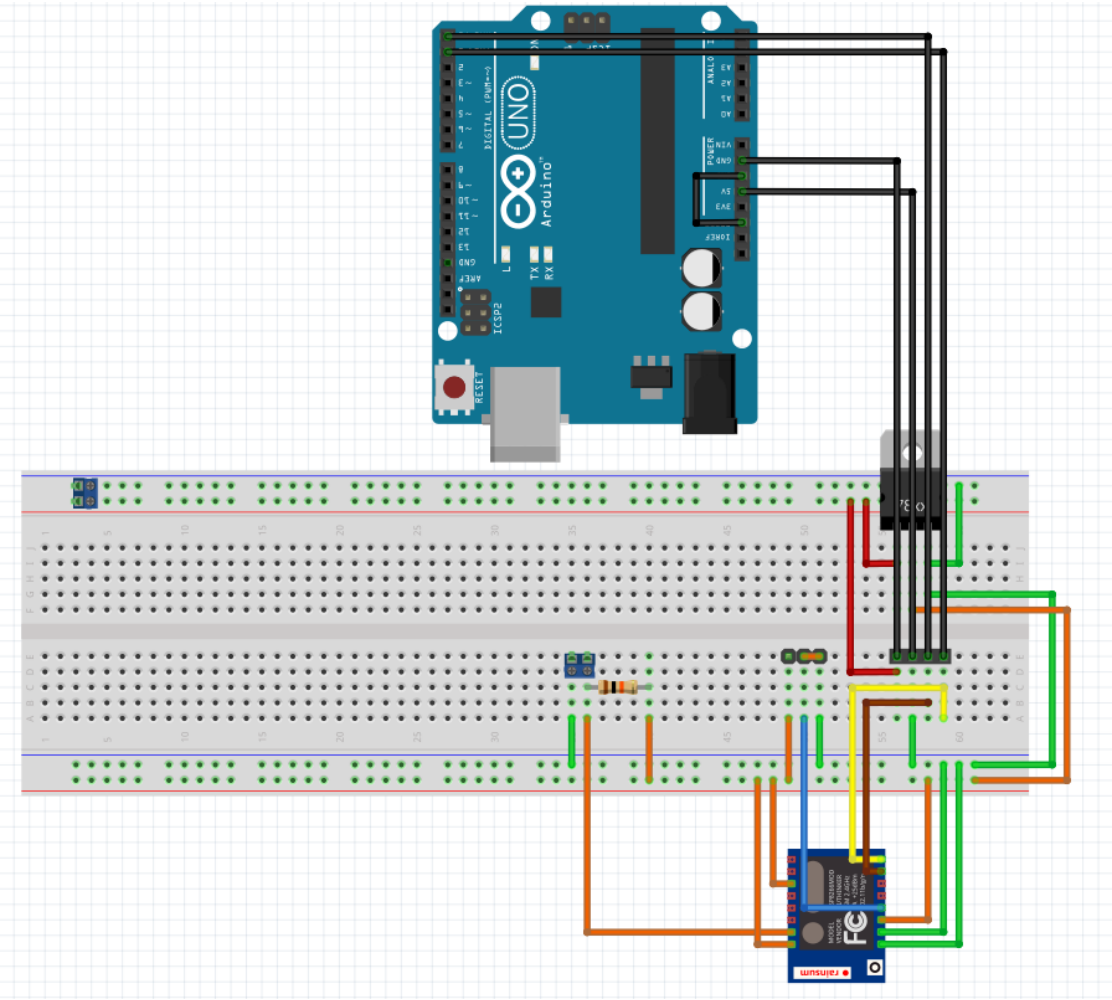

Figure 1. Door sensor IoT physical device's breadboard scheme

With this cabling done, Arduino UNO device is connected to computer using USB cable. Computer is running Arduino IDE software that allows to flash the code for ESP8266 wireless module on PCB. The code contains the information for WLAN, MQTT broker, local server, and virtual IoT device of the

custom SMART house solution. The code for flashing PCB of Door Sensor is displayed in appendix 6. With code flashed device is physically disconnected from computer and Arduino UNO device. Then a jumper switch on PCB is manually switched from PGM to RUN mode. Finally, the IoT physical device is put into enclosure and installed in specific location for its functionality. Device is powered on and tested using MQTT.fx software. A specific component for device is the magnetic read sensor. The magnetic read sensor contains two magnets of which one is connected to PCB using copper wire. When magnets are separated, MQTT message is sent to the local server. Message contains OPEN or CLOSED strings of text.

A virtual IoT thing is created and named as SecuritySystem in the home.items file. This is a switch type item. Its' purpose is to enable one of automation rules. Another IoT virtual thing is created and named as GPDoorSensor1. This is a contact type item that communicates with IoT physical device. A picture of door is coded to appear in the user interface. For users item name that is shown in the sitemap is coded to be Bedroom. Next to the name bedroom a state of IoT physical device is coded to be displayed, which changes between CLOSED and OPEN. Change of state is received by MQTT message from IoT physical device configured and powered on earlier. MQTT broker path, that was specified while flashing the IoT physical device, is used for IoT virtual thing. The code of SecuritySystem is shown in Code 1. The code for automation rule of the bedroom door sensor is shown in Code 2.

```
//Security system
Switch SecuritySystem "Security System"
Contact GPDoorSensor1 "Bedroom [%s]" <door> {mqtt="<[broker:GP-SmartHouse/security/GP-DoorSensor1:state:default]"}
```

Code 1. IoT virtual thing SecuritySystem

```
//Bedroom doors
rule "Bedroom Door Sensor"
when
        Item GPDoorSensor1 received update OPEN
then
        if(SecuritySystem.state == ON)
        {
                sendBroadcastNotification("SECURITY SYSTEM: BEDROOM DOOR OPEN")
        }
end
```

Code 2. Automation rule of Bedroom Door Sensor

The automation rule named Bedroom Door Sensor is created at home.rules file. The rule is enabled by earlier created switch SecuritySystem by using if

condition. If switch state is on the rule is enabled, if off – the rule is disabled. The rule itself states when item GPDoorSensor1 receives an update OPEN from IoT physical Door Sensor device, a broadcast notification is sent. The code of the Bedroom Door Sensor rule.

### 3.5.2. LED light

To configure LED Light IoT physical device for this custom smart house solution device is prepared and binded with created IoT virtual things. First physical IoT device configurations are performed. A jumper switch on PCB is manually switched from RUN to PGM mode. Then LED Light physical IoT device is connected to Arduino UNO device using Male-to-Male cables. Then Arduino UNO Power section is cabled from RES to GND using Male-to-Male cable. Figure 2 displays LED light physical IoT device's breadboard scheme connected to Arduino UNO device.

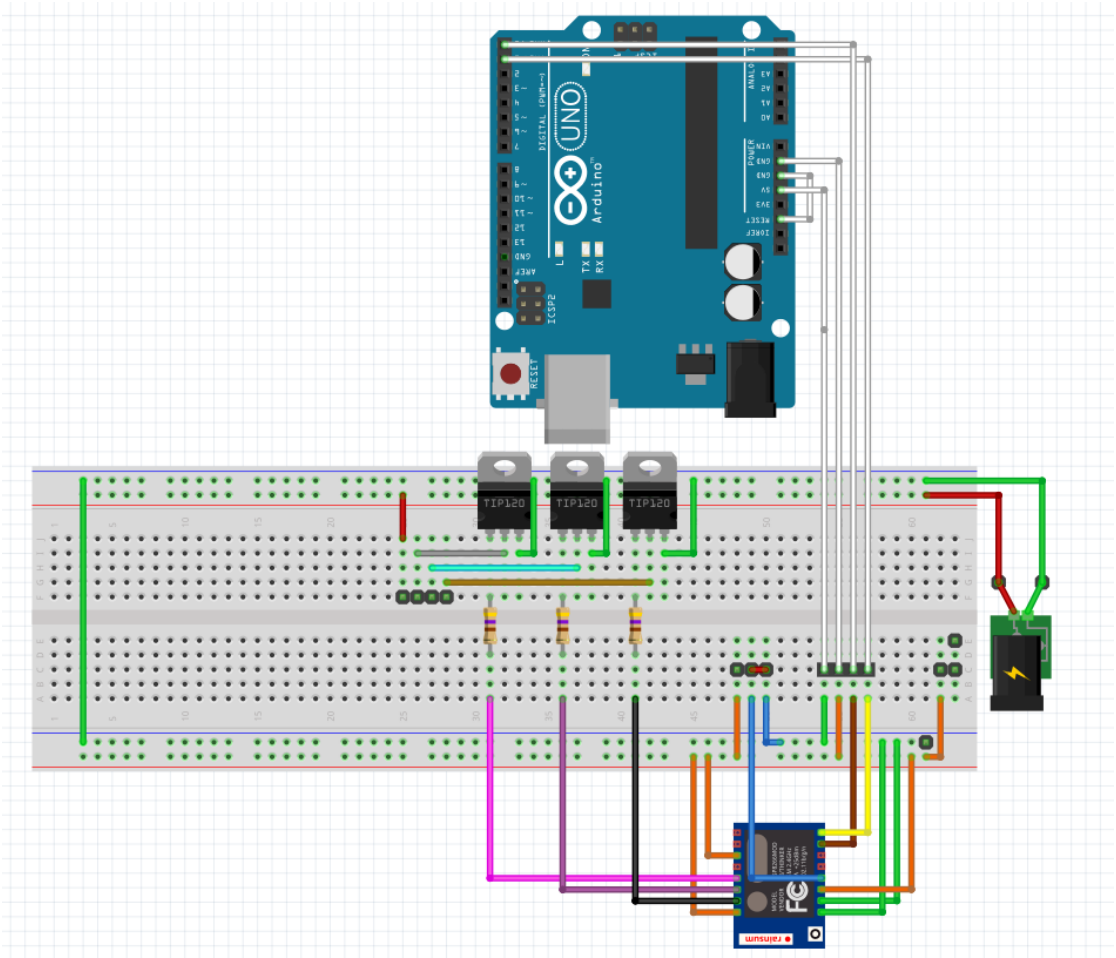


Figure 2. LED Light IoT physical device's breadboard scheme

With this cabling completed, Arduino UNO device is connected to computer using USB cable. The computer is running Arduino IDE software that allows to flash the code for ESP8266 wireless module on PCB the same way like Door sensor IoT physical device. The code contains the information for WLAN, MQTT broker, the local server, and IoT virtual device of this smart house solution. The code for flashing PCB of LED Light is displayed in appendix 7.

With code flashed device is physically disconnected from computer and Arduino UNO device. Then a jumper switch on PCB is manually switched from PGM to RUN mode. Finally, the IoT physical device is put into enclosure and installed in specific location for its functionality. Device is powered on and tested using MQTT.fx software. A specific component for device is the LED light strip. The LED light strip contains 5 meters of LEDs that are connected to PCB using 4 male-to-female connectors. When light needs to be turned on, MQTT message is sent from the local server. Message contains values for the LED light colour in format of X:Y:Z, where X is red colour value, Y is green colour value, Z is blue colour value. For example, 100:100:100 results white colour LED light.

An IoT virtual thing LEDStripControl1Group is created at the home.items file. This is a group type item. Its purpose is to collect LED Light items into a group. Another IoT virtual thing is created and named LEDStripControl1Color. This is a colour type item. It allows user to choose value for LED light using colour palette instead of manually writing strings of values. This item is sends these selected values for LEDStripControl1Group thing. Finally, the LEDStripControl1String IoT virtual device is created. This item takes values from LEDStripControl1Group thing. These values are sent as single string to the MQTT broker address that was flashed into physical LED Light IoT physical device. The code of LEDStripControl1Group, LEDStripControl1Color and LEDStripControl1String things are displayed in code 3. The variables for automation rule LED light is are displayed in code 4. The automation rule LED light is displayed in code 5.

```
//LED light
Group GPLEDStripControl1Group "LED Light" (All)
Color GPLEDStripControl1Color "LED Light" <colorpicker> (GPLEDStripControl1Group)
String GPLEDStripControl1String (LEDStripControl1Group) {mqtt=">[broker:GP-SmartHouse/lights/GP-LEDStripControl1:command:*:default]"}
```

Code 3. LEDStripControl1Group, LEDStripControl1Color and LEDStripControl1String IoT virtual things

```
var HSBType hsbValue
var int redValue
var int greenValue
var int blueValue
var String RGBvalues
```

Code 4. Variables for automation rules LED Light

```
//LED light
rule "LED Light"
when
        Item GPLEDStripControl1Color changed
then
        hsbValue = GPLEDStripControl1Color.state as HSBType
        redValue = hsbValue.red.intValue
        greenValue = hsbValue.green.intValue
        blueValue = hsbValue.blue.intValue
        RGBvalues = redValue.toString + ";" + greenValue.toString + ";" + blueValue.toString + ";"
        sendCommand(GPLEDStripControl1String, RGBvalues)
        logInfo("GPLEDStripControl1Color", RGBvalues)
end
```

Code 5. Automation rule LED Light

To make LED light IoT physical device work automation rule named LED Light is created at home.rules file. The rule itself states when IoT virtual thing LEDStripControl1Color value is changed (for example, using colour picker), RGB values are sent as command to the LEDStripControl1String thing, that sends MQTT message to LED Light IoT physical device.

### 3.5.3. Fire Sensor

To configure Fire Sensor IoT physical device for custom smart house solution device is prepared and binded with created IoT virtual thing. First IoT physical device configurations are performed. A jumper switch on PCB is manually switched from RUN to PGM mode. Then Fire sensor IoT physical device is connected to Arduino UNO device using Male-to-Male cables. Then Arduino UNO Power section is cabled Male-to-Male from RES to GND. Figure 3 displays Fire Sensor IoT physical device's breadboard scheme connected to Arduino UNO device.

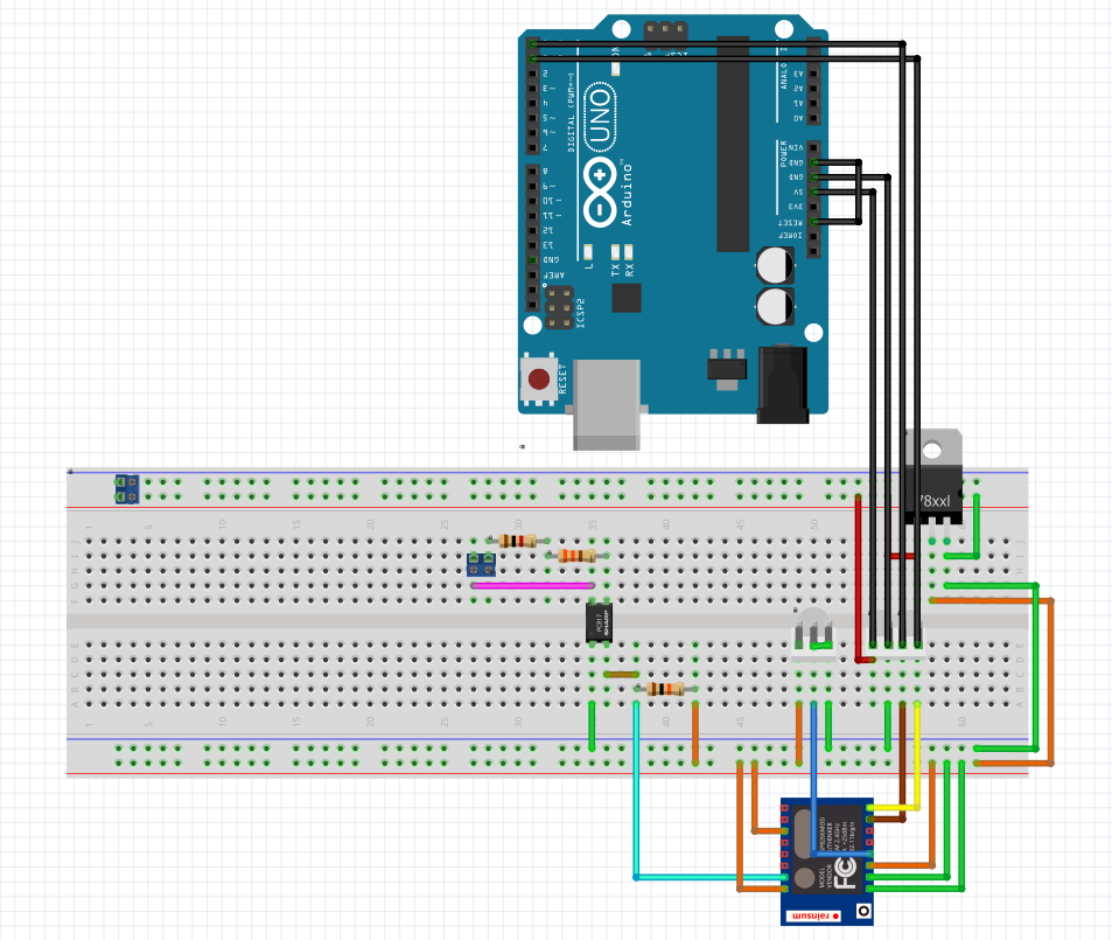


Figure 3. Fire Sensor IoT physical device's breadboard scheme

With this cabling completed, Arduino UNO device is connected to computer using USB cable. The computer is running Arduino IDE software that allows to flash the code for ESP8266 wireless module on PCB the same way like Door sensor or LED Light IoT physical devices. The code contains the information for WLAN, MQTT broker, local server, and virtual IoT device of the custom smart house solution. The code for flashing PCB of Fire sensor is displayed in appendix 8.

With code flashed device is physically disconnected from computer and Arduino UNO device. Then a jumper switch on PCB is manually switched from PGM to RUN mode. Finally, the IoT physical device is put into enclosure. Device is powered on and tested using MQTT.fx software and a battery to simulate fire detector alarm. A specific component for device is two copper cables. These cables are used to connect to fire detector device that support interconnect feature. When interconnected fire detector sends an alarm, electrical signal is sent to Fire sensor IoT physical device, which then sends

MQTT message to the local server. Message contains string of text that can either be No fire, or FIRE DETECTED.

An IoT virtual thing FireSensor1 is created at the home.items file. This is a string type item. Its purpose is to display a string received by the Fire detector IoT physical device. For users item name that is shown in the sitemap is coded to be Home for location of Fire Sensor IoT physical device. Next to the name Home a string received from Fire sensor physical IoT device is displayed, which changes between No fire and FIRE DETECTED. The code of FireSensor1 is displayed in Code 6. The rule Fire alarm is displayed in Code 7.

```
//Fire alarm
String GPFireSensor1 "Home [%s]" <fire> {mqtt="<[broker:GP-SmartHouse/utilities/GP-FireSensor1:state:default]"}
```

Code 6. FireSensor1 IoT virtual thing

```
//Fire alarm
rule "Fire detector"
when
        Item GPFireSensor1 received update
then
        if(GPFireSensor1.state == "FIRE DETECTED!")
        {
                sendBroadcastNotification("FIRE SENSOR: FIRE DETECTED!!!")
        }
end
```

Code 7. The code of automation rule Fire alarm

Since fire is dangerous a rule “Fire detector” is created at “home.rules” file. The rule itself states when “Item” “FireSensor1” string receives an update from “Fire Sensor” physical IoT device, it sends a broadcast notification stating "FIRE SENSOR: FIRE DETECTED!!!".

## 3.6. Thesis project overview

Smart house products are expensive to implement for average users. Some products are tied to vendors, making expansion for the system even more expensive. In some cases, if anything breaks, not only the users have to pay for the fix, but for technical support as well. This project plan is created to find out how smart house solution prices are calculated in the market.

First, to find out how expensive this custom smart house solution is, the price for the administrator's work and prices for physical devices are calculated. The person who sets up the solution is the network administrator. That means that he is paid EUR 20.62 per hour. If administrator is working overtime, his wage increases to EUR 41.24 per hour. Device prices are simply added to the resource pool. Without the administrator's work included, the price of this custom smart house solution is EUR 175.40. Table 4 displays the list of resources and their prices.

Table 4. List of the project's resources

| **Resource Name** | **Type** | **Std. Rate EUR** | **Ovt. Rate EUR** | **Cost/Use EUR** |
|---|---|---|---|---|
| Gintaras Pacevičius | Work | 20.62/hr | 41.24/hr | 0.00 |
| Raspberry Pi 3 Model B | Material | - | - | 34.75 |
| DCCduino UNO R3 with (USB A/B cable) | Material | - | - | 7.90 |
| Edimax AC750 router | Material | - | - | 39.99 |
| Raspberry Pi 3 heatsink, case | Material | - | - | 9.99 |
| SanDisk Ultra 16 Gb SD card | Material | - | - | 11.99 |
| "Door sensor" | Material | - | - | 15.99 |
| "LED Light" | Material | - | - | 15.99 |
| "Fire Sensor" | Material | - | - | 7.99 |
| LED light strip (5 meters) | Material | - | - | 15.99 |
| 4x S/FTP CAT6 LAN cables | Material | - | - | 1.56 |
| M-M cable pack (100 cables) | Material | - | - | 3.19 |
| 3x Power adapters (Specification 5.1V 2.5A) | Material | - | - | 17.97 |

To include the administrator's work price, the project lifetime is calculated. The project has three main phases. These phases are project planning, project implementation, and project documentation. Planning includes finding out software, hardware requirements, performing network design, and it takes 15 work days to complete. The project implementation includes network configuration, the local server's and IoT devices' configurations, and it takes 7 work days to complete. The project documentation takes 5 work days to complete. With the project duration known, the administrator must perform 392 hours of work. This results to a sum of EUR 8083 for the administrator's work. This means that if user buys this custom smart house solution as a product, and wishes that the administrator plans it, implements it, and documents it for him, the project would cost the total of EUR 8258. Table 5 displays project lifetime.

Table 5. Project lifetime

| Task Name | Duration | Start | Finish | Resource Names |
|---|---|---|---|---|
| **Software requirements** | 5 days | 3/18/19 | 3/22/19 | Gintaras Pacevičius |
| **Hardware requirements** | 5 days | 3/25/19 | 3/29/19 | Gintaras Pacevičius |
| **Network design** | 5 days | 4/1/19 | 4/5/19 | Gintaras Pacevičius |
| **Network configuration** | 2 days | 4/8/19 | 4/9/19 | Gintaras Pacevičius |
| **Local server configuration** | 2 days | 4/10/19 | 4/11/19 | Gintaras Pacevičius |
| **IoT devices configurations** | 3 days | 4/12/19 | 4/16/19 | Gintaras Pacevičius |
| **Project documentation** | 5 days | 4/17/19 | 4/23/19 | Gintaras Pacevičius |

If the project owner wants to sell the whole project for the price other taxes must also be calculated. According to Wikipedia (2019) the profit tax in Finland is 20% and value-added tax is 23% (Wikipedia 2019). The project owner adds 10% markup price to make a profit. This affects equipment price, that increases to EUR 192.94 and human resource price that raises to EUR 9,066.74 resulting to the total sum of EUR 9,259.68 for this custom smart

house solution. To sell the whole project (for another company, etc.) 100% markup price is added resulting to a total of EUR 18,519.36. A value-added tax is calculated and added to the total of EUR 22,778.81. That means that the project profit without VAT is EUR 9,259.68. To get the net profit, profit tax must be paid. This results in EUR 7,407.74 net profit for the project's owner. Appendix 9 displays the calculations proper price of the custom smart house solutionof the thesis custom smart house solution project price.

According to loxone.com (2019), a budget type of smart house solution offered by them would cost EUR 5324 (Loxone 2019). Medium type smart house solution would cost EUR 10648. And finally, the smart house with all the features included would cost EUR 15972. Knowing that, the thesis solution price is slighly below the medium type of smart house product in the market. However, the thesis solution is scalable, users have full control of it, and do not owe any more additional fees such as technical support, etc. If the users decide to do the all administrator work themsevles, the price for the solution drops to EUR 192.94 (for devices only). This is only 2.33% of the whole solution's price. This is an example that displays that users who buy the solution are paying mostly for the work performed by the specialists, rather than for the solution itself. By building and designing the smart house solutions themselves, users saves a lot of money.

# 4. RESULTS

This chapter focuses on the results of this custom built smart house solution. The chapter contains information about how the smart house market problems such as scalability, flexibility, security, price and complexity are addressed during this thesis project implementation.

Since the solution uses OpenHABian as the core operating system, there are not many scalability problems. The solution scales well, because there is no limit to the number of IoT devices that can be added to the system. This is due to the fact that any IoT physical or virtual thing added using lines of code does not require official vendor support. Currently, the network for IoT devices is designed to support 254 hosts. However, if more host addresses are required, network can be reconfigured to support up to 16 777 214 hosts by simply

using the private address range of 10.0.0.0 network with 255.0.0.0 subnet mask. The only problem this much flexibility introduces is more complexity. To add all of those IoT devices a lot of manual work is required.

Similarly to scalability so is flexibility. Using OpenHABian, there are two ways of adding new devices. It is possible to add physical and virtual IoT devices either using the Paper user interface, or lines of code. One way added device can be modified the other way. For example, Door sensor added using lines of code can be edited using the Paper user interface. There are no strict rules for what IoT devices should be, since there are no requirements for official vendor drivers, support, etc. PCBs are used as physical IoT devices throughout the project. To integrate such devices into the custom smart house solution MQTT protocol is used for communication. There might be a number of other protocols that could also be used. Automation is also coded. There are no strict rules for automation coding, except for grammar. That means that user can automate everything writing in his own style, using his own variables, using whatever approach he wishes. In exchange of more complex manual work solution offers more flexibility.

Since this custom smart house solution is designed to have no physical security controls and no vital systems attached to it, the solution itself is secure. The solution network, on the other hand, relies heavily on wireless technology. All networks can be breached. Wireless technology introduces additional attack vectors. In case attackers manage to gain access to the solution network, the gains for him would be minimal. This is because the solution network is isolated from other networks. No sensitive data could be stolen from users' network, because there are no devices with sensitive data connected to the smart house network. No physical security could be manipulated by attackers, because there are no physical security controls in the solution. No vital system could be affected, because there are none connected. This custom smart house solution can be even more secure, if administrators would implement more complex security features such as VPN, port-security, etc.

Since all the work is performed by the owner of the system, the price for this custom smart house system drops to only the price for devices. Users only

need to buy components and essential hardware. This means that the solution itself is very cheap, compared to the ones that companies sell. Because owner is fully responsible for maintaining the solution up and running, some problems could get complex to solve.

The biggest problem for this custom smart house solution is complexity. To keep the solution running, all modifications and configurations require manual work. Manual work requires users' time and specific knowledge that they may not have. Not many home users are willing to spend their free time learning information technology. Complexity is the price to pay for a cheap, secure, flexible, and scalable custom smart house solution.

## 5. CONCLUSION

There are many aspects of home life that could be improved, or even completely automated, using smart house technologies. While people argue that solutions offered by the companies are too expensive and insecure, there are other people who test out their own custom solutions, which proves quite opposite truths. Custom solutions created by the users are their own personal life improvements. If users can think of logical conditions required for process to occur, they can automate, or enhance the whole process using electronics. This thesis goal was for the users to build a custom smart house solution that is cheap, flexible, scalable, secure and simple. The project results proved that in exchange of a more complex approach, it is possible for users to build their own smart house solutions. Technologies used during the thesis project are accessible for any users that would like to build a custom solution. People who are willing to learn smart house-related technologies, and are willing to overcome complexity-related problems, will improve their life with automation and smart appliances. So while working harder might pay off faster, in general, working smarter, not harder, might benefit users for the long run.

REFERENCES

Brice A. 2018. SmartHome University. Best of open source smart home: Home Assistant vs OpenHAB. WWW article. Available at: https://smarthome.university/your-smart-home-platform-home-assistant-vs-openhab/ [Accessed 19 March 2019].

Cisco Networking Academy. 2019. CCNA Security. Implementing Virtual Private Networks. Introducing VPNs. WWW document. Available at: https://static-course-assets.s3.amazonaws.com/CCNAS2/en/index.html#8.1.1.1 [Accessed 17 March 2019].

Cisco Networking Academy. 2019. CCNA1. Introduction to networks. Communication Fundamentals. WWW document. Available at: https://static-course-assets.s3.amazonaws.com/ITN6/en/index.html#3.1.1.1 [Accessed 14 March 2019].

Cisco Networking Academy. 2019. CCNA1. Introduction to networks. Overview to network components. WWW document. Available at: https://static-course-assets.s3.amazonaws.com/ITN6/en/index.html#1.2.1.1 [Accessed 14 March 2019].

Electrical4U. 2019. What is Printed Circuit Board or PCB? WWW article. Available at: https://www.electrical4u.com/what-is-printed-circuit-board-or-pcb/ [Accessed 17 March 2019].

Florida Center for Instructional Technology. 2013. What is a Network? WWW publication. Available at: http://fcit.usf.edu/network/chap1/chap1.htm#LocalAreaNetwork [Accessed 14 March 2019].

Francoise C. 2018. Steemit. Benefits of Smart Home Technology. WWW article. Available at: https://steemit.com/dxchain/@francoisclaude/benefits-of-smart-home-technology [Accessed 19 March 2019].

Hendricks D. 2014. IoT Evolution. The History of Smart Homes. WWW article. Available at: https://www.iotevolutionworld.com/m2m/articles/376816-history-smart-homes.htm [Accessed 19 February 2019].

Hoffman C. 2018. How-to Geek. What is a VPN, and why would I need one? WWW article. Available at: https://www.howtogeek.com/133680/htg-explains-what-is-a-vpn/ [Accessed 17 March 2019].

Home Assistant Dev Docs. 2019. Components architecture. WWW document. Available at: https://developers.home-assistant.io/docs/en/architecture_components.html [Accessed 19 March 2019].

International Organization for Standardization. 2016. ISO/IEC 20922:2016. Information technology -- Message Queuing Telemetry Transport (MQTT). WWW document. Available at: https://www.iso.org/standard/69466.html [Accessed 17 March 2019].

International Telecommunication Union (ITU). 2012. ITU-T Y.4000/Y.2060. WWW document. Available at: https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060 [Accessed 17 March 2019].

Loxone. 360° Home Automation The Complete Solution. WWW document. Available at: https://www.loxone.com/enus/ [Accessed 17 March 2019].

Marshall B., Wilson T., Johnson B. 2019. HowStuffWorks. How WiFi Works. WWW document. Available at: https://computer.howstuffworks.com/wireless-network.htm [Accessed 14 May 2019].

Mitchell B. 2019. Lifewire. What is a LAN (Local Area Network)? An introduction to the essential concepts of a LAN. WWW article. Available at: https://www.lifewire.com/local-area-network-816382 [Accessed 14 March 2019].

OpenHAB. 2019. OpenHABian – hassle-free openHAB setup. WWW document. Available at: https://www.openhab.org/docs/installation/openhabian.html [Accessed 19 March 2019].

Rouse M. 2019. Internet of Things Agenda. Smart home or building (home automation and domotics). WWW article. Available at: https://internetofthingsagenda.techtarget.com/definition/smart-home-or-building [Accessed 19 February 2019].

Smart Home Blog. 2018. What is MQTT and How to use it with OpenHab? WWW document. Available at: https://www.smarthomeblog.net/mqtt-openhab/ [Accessed 17 March 2019].

Techopedia. 2019. Single-Board computer (SBC). WWW article. Available at: https://www.techopedia.com/definition/9266/single-board-computer-sbc [Accessed 17 March 2019].

Wikipedia. 2019. Corporate tax. WWW document. Available at: https://en.wikipedia.org/wiki/Corporate_tax [Accessed 14 April 2019].

Wikipedia. 2019. Value added tax. WWW document. Available at: https://en.wikipedia.org/wiki/Value-added_tax [Accessed 14 April 2019].

Yi Man Li R., Ching Yu Li H., Kei Mak C., Beigi Tang T. 2016. International journal of smart home. Sustainable smart home and home automation: big data analytics approach. WWW document. Available at: https://pdfs.semanticscholar.org/07d2/e25c97ef935a80811443da65a5ead545539e.pdf?_ga=2.139026611.1805603897.1546310064-1041290255.1543912813 [Accessed 17 March 2019].

# APPENDICES

## Appendix 1. “home.things” file content

```
//This is the things file
```

Appendix 2. “home.items” file content

```
//This is Item file

//Security system
Switch SecuritySystem "Security System"
Contact GPDoorSensor1 "Bedroom [%s]" <door> {mqtt="<[broker:GP-
SmartHouse/security/GP-DoorSensor1:state:default]" }

//LED light
Group GPLEDStripControl1Group "LED Light" (All)
Color GPLEDStripControl1Color "LED Light" <colorpicker>
(GPLEDStripControl1Group)
String GPLEDStripControl1String (LEDStripControl1Group)
{mqtt=">[broker:GP-SmartHouse/lights/GP-
LEDStripControl1:command:*:default]"}

//Fire alarm
String GPFireSensor1 "Home [%s]" <fire> {mqtt="<[broker:GP-
SmartHouse/utilities/GP-FireSensor1:state:default]"}
```

Appendix 3. “home.sitemap” file content

```
sitemap home label="Smart Home System"
{
    Frame label="Security"
    {
        Switch item=SecuritySystem
        Text item=GPDoorSensor1
    }
    Frame label="LED Light"
    {
        Colorpicker item=GPLEDStripControl1Color
    }
    Frame label="Fire Detector"
    {
        Text item=GPFireSensor1
    }
}
```

Appendix 4. “home.rules” file content

```
var HSBType hsbValue
var int redValue
var int greenValue
var int blueValue
var String RGBvalues

//This is rule file

//Bedroom doors
rule "Bedroom Door Sensor"
when
    Item GPDoorSensor1 received update OPEN
then
    if(SecuritySystem.state == ON)
    {
        sendBroadcastNotification("SECURITY SYSTEM: BEDROOM DOOR
OPEN")
    }
end

//LED light
rule "LED Light"
when
    Item GPLEDStripControl1Color changed
then
    hsbValue = GPLEDStripControl1Color.state as HSBType
    redValue = hsbValue.red.intValue
    greenValue = hsbValue.green.intValue
    blueValue = hsbValue.blue.intValue
    RGBvalues = redValue.toString + ";" + greenValue.toString + ";” +
blueValue.toString + ";
```

```
    sendCommand(GPLEDStripControl1String, RGBvalues)
    logInfo("GPLEDStripControl1Color", RGBvalues)
end

//Fire alarm
rule "Fire detector"
when
    Item GPFireSensor1 received update
then
    if(GPFireSensor1.state == "APTIKTA UGNIS!")
    {
        sendBroadcastNotification("FIRE SENSOR: FIRE DETECTED!!!")
    }
End
```

# Appendix 5. User interface “Basic UI” view

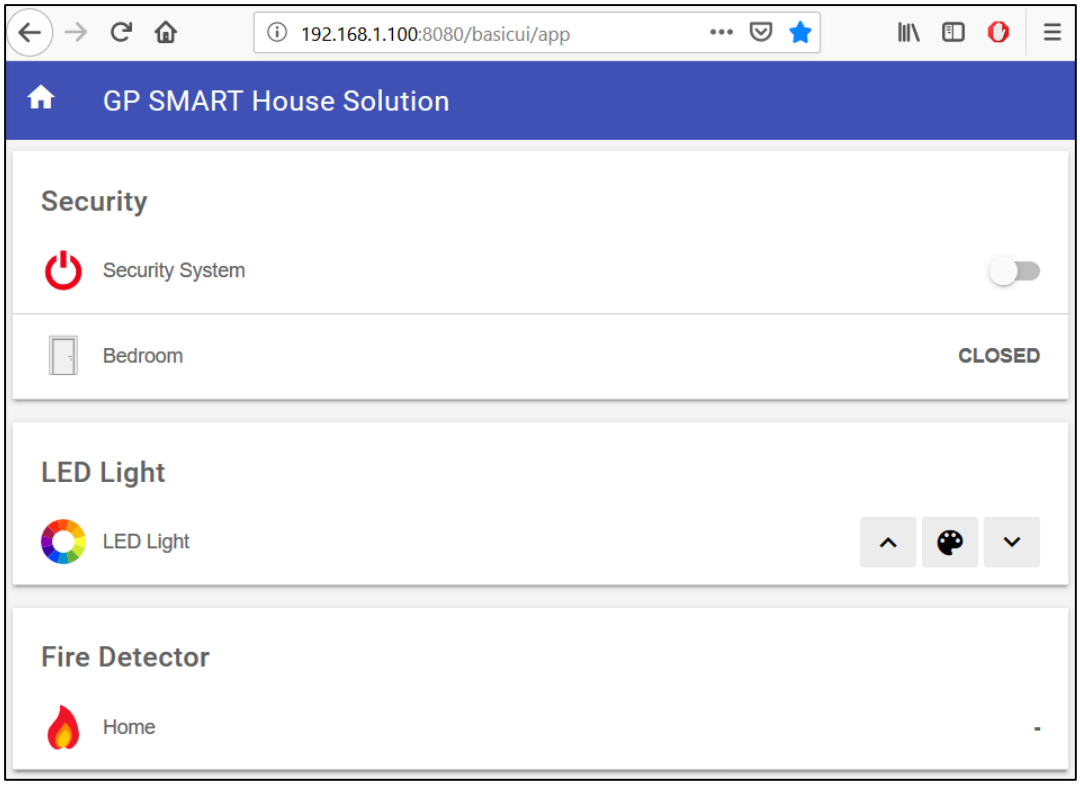


Figure 4. User interface ”Basic UI” view using WEB browser

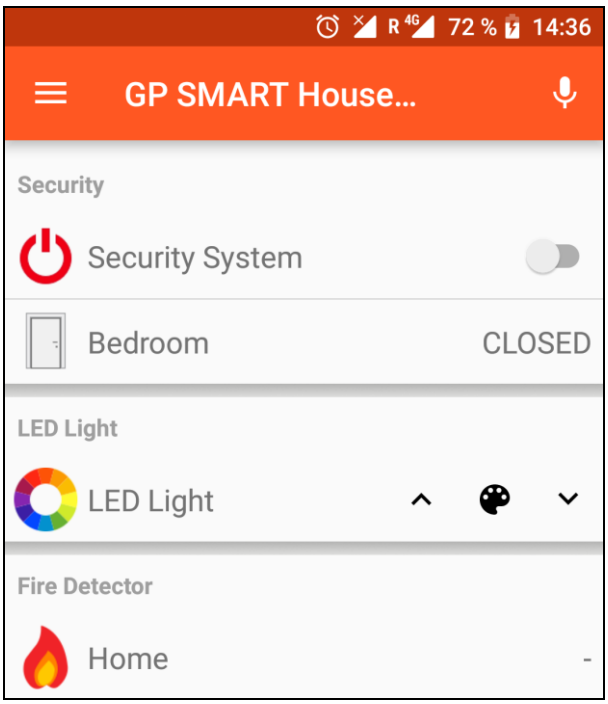


Figure 5. User interface ”Basic UI” view using android application.

Appendix 6. Code flashed in “Door Sensor” Physical IoT device

```
#include <ESP8266WiFi.h>
#include <MQTTClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <ESP8266HTTPUpdateServer.h>
const char* ssid = "Namai-Wifi";
const char* password = "61911741";
const char* host = "GP-DoorSensor1";
const char* update_path = "/firmware";
const char* update_username = "admin";
const char* update_password = "admin";
char* outTopic = "GP-SmartHouse/security/GP-DoorSensor1";
const char* server = "192.168.1.100";
const char* mqttDeviceID = "GP-SmartHouseDevice1";
long unsigned int lowIn;
long unsigned int pause = 100;
boolean lockLow = true;
boolean takeLowTime;
int sensorPin = 13;
ESP8266WebServer httpServer(80);
ESP8266HTTPUpdateServer httpUpdater;
WiFiClient net;
MQTTClient client;
unsigned long lastMillis = 0;
void connect();
void setup()
{
  pinMode(sensorPin, INPUT);
  digitalWrite(sensorPin, LOW);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  client.begin(server, net);
  client.onMessage(messageReceived);
```

```
  connect();
  MDNS.begin(host);
  httpUpdater.setup(&httpServer, update_path, update_username,
update_password);
  httpServer.begin();
  MDNS.addService("http", "tcp", 80);
}
void connect()
{
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(1000);
  }
  while (!client.connect(mqttDeviceID))
  {
    delay(1000);
  }
}
void loop()
{
  client.loop();
  delay(10);
  if(!client.connected())
  {
    connect();
  }
  httpServer.handleClient();
  if(digitalRead(sensorPin) == HIGH)
  {
    if(lockLow)
    {
      lockLow = false;
      client.publish(outTopic, "OPEN");
      delay(50);
```

```
    }
    takeLowTime = true;
  }
  if(digitalRead(sensorPin) == LOW)
  {
    if(takeLowTime)
    {
      lowIn = millis();
      takeLowTime = false;
    }
    if(!lockLow && millis() - lowIn > pause)
    {
      lockLow = true;
      client.publish(outTopic, "CLOSED");
      delay(50);
    }
  }
}
void messageReceived(String &topic, String &payload)
{
}
```

Appendix 7. Code flashed in “LED Light” Physical IoT device

```
#include <ESP8266WiFi.h>
#include <MQTTClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <ESP8266HTTPUpdateServer.h>
const char* ssid = "Namai-Wifi";
const char* password = "61911741";
const char* host = "GP-LEDStripControl1";
const char* update_path = "/firmware";
const char* update_username = "admin";
const char* update_password = "Admin";
char* subscribeTopic = "GP-SmartHouse/lights/GP-LEDStripControl1";
const char* server = "192.168.1.100";
const char* mqttDeviceID = "GP-SmartHouseDevice2";
const int greenLed = 12;
const int redLed = 14;
const int blueLed = 13;
String greenLedVal = "0";
String redLedVal = "0";
String blueLedVal = "0";
String greenLedValLast = "0";
String redLedValLast = "0";
String blueLedValLast = "0";
ESP8266WebServer httpServer(80);
ESP8266HTTPUpdateServer httpUpdater;
WiFiClient net;
MQTTClient client;
unsigned long lastMillis = 0;
void connect();
```

```
void setup()
{
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  client.begin(server, net);
  client.onMessage(messageReceived);
  connect();
  MDNS.begin(host);
  httpUpdater.setup(&httpServer, update_path, update_username,
update_password);
  httpServer.begin();
  MDNS.addService("http", "tcp", 80);
}
void connect()
{
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(1000);
  }
  while (!client.connect(mqttDeviceID))
  {
    delay(1000);
  }
  client.subscribe(subscribeTopic);
}
void loop()
{
  client.loop();
  delay(10);
  if(!client.connected())
  {
    connect();
  }
  httpServer.handleClient();
```

```
}
void messageReceived(String &topic, String &payload)
{
  String stringOne = payload;
  Serial.println(stringOne);
  int firstClosingBracket = stringOne.indexOf(';')+1;
  int secondOpeningBracket = firstClosingBracket + 1;
  int secondClosingBracket = stringOne.indexOf(';', secondOpeningBracket);
  int thirdOpeningBracket = secondClosingBracket + 1;
  int thirdClosingBracket = stringOne.indexOf(';', thirdOpeningBracket);
  greenLedVal = stringOne.substring(0 , (firstClosingBracket - 1));
  redLedVal = stringOne.substring(firstClosingBracket ,
secondClosingBracket);
  blueLedVal = stringOne.substring((secondClosingBracket +1) ,
thirdClosingBracket);
  if ((blueLedVal != blueLedValLast) || (greenLedVal != greenLedValLast) ||
(redLedVal != redLedValLast))
  {
  Serial.println(blueLedVal.toInt());
  Serial.println(redLedVal.toInt());
  Serial.println(greenLedVal.toInt());
  analogWrite(blueLed, ((blueLedVal.toInt()) * 10.23));
  analogWrite(redLed, ((redLedVal.toInt()) * 10.23));
  analogWrite(greenLed, ((greenLedVal.toInt()) * 10.23));
  greenLedValLast = greenLedVal;
  redLedValLast = redLedVal;
  blueLedValLast = blueLedVal;
  }
  else
  {
  }
}
```

Appendix 8. Code flashed in “Fire Sensor” Physical IoT device

```
#include <ESP8266WiFi.h>
#include <MQTTClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <ESP8266HTTPUpdateServer.h>
const char* ssid = "Namai-Wifi";
const char* password = "61911741";
const char* host = "GP-FireSensor1";
const char* update_path = "/firmware";
const char* update_username = "admin";
const char* update_password = "Admin";
char* outTopic = "GP-SmartHouse/utilities/GP-FireSensor1";
const char* server = "192.168.1.100";
const char* mqttDeviceID = "GP-SmartHouseDevice3";
int calibrationTime = 10;
long unsigned int lowIn;
long unsigned int pause = 100;
boolean lockLow = true;
boolean takeLowTime;
int pirPin = 13;
ESP8266WebServer httpServer(80);
ESP8266HTTPUpdateServer httpUpdater;
WiFiClient net;
MQTTClient client;
unsigned long lastMillis = 0;
void connect();
void setup()
{
  pinMode(pirPin, INPUT);
  digitalWrite(pirPin, LOW);
  WiFi.mode(WIFI_STA);
```

```
  WiFi.begin(ssid, password);
  client.begin(server, net);
  client.onMessage(messageReceived);
  connect();
  MDNS.begin(host);
  httpUpdater.setup(&httpServer, update_path, update_username,
update_password);
  httpServer.begin();
  MDNS.addService("http", "tcp", 80);
}
void connect()
{
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(1000);
  }
  while (!client.connect(mqttDeviceID))
  {
    delay(1000);
  }
}
void loop()
{
  client.loop();
  delay(10);
  if(!client.connected())
  {
    connect();
  }
  httpServer.handleClient();
  if(digitalRead(pirPin) == HIGH)
  {
    if(lockLow)
```

```
    {
      lockLow = false;
      client.publish(outTopic, "No fire");
      delay(50);
    }
    takeLowTime = true;
  }
  if(digitalRead(pirPin) == LOW)
  {
    if(takeLowTime)
    {
      lowIn = millis();
      takeLowTime = false;
    }
    if(!lockLow && millis() - lowIn > pause)
    {
      lockLow = true;
      client.publish(outTopic, "FIRE DETECTED!");
      delay(50);
    }
  }
}
void messageReceived(String &topic, String &payload)
{
}
```

Appendix 9. Project price of this custom smart house solution

Table 6. Project price calculations including taxes and markup price

| **Custom smart house solution implementation cost** | **Price, EUR** | **Markup, 10%, EUR** | **Total price, EUR** |
|---|---|---|---|
| Technical equipment | € 175.40 | € 17.54 | € 192.94 |
| Human resource price | € 8,083.04 | € 808.30 | € 9,066.74 |
| | | Total | € 9,259.68 |
| | | Markup, 100% | € 9,259.68 |
| | | Sum with markup | € 18,519.36 |
| | | Value-added tax, 23% | € 4,259.45 |
| | | Total price including VAT | € 22,778.81 |

| 9,259.68 | = | 18,519.36 | - | 9,259.68 |
|---|---|---|---|---|
| Project profit, EUR | | Project selling price, without VAT, EUR | | Project implementation price, EUR |

| 7,407.74 | = | 9,259.68 | - | 1,851.94 |
|---|---|---|---|---|
| Net profit, EUR | | Project profit, EUR | | Profit tax (20%), EUR |