UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Enrih Sinilaid

Monitoring and controlling smart home appliances using IoT devices

Bachelor's Thesis (9 ECTS)

Supervisor: Chinmaya Kumar Dehury

Tartu 2021

Monitoring and controlling smart home appliances using IoT devices

Abstract:

Usage of different smart home appliances and systems is becoming increasingly more popular in many households. There are many key points for rising number of adopters. For some it is the price as these systems and appliances are not that new to the market anymore and thus are more reasonably priced. Also, with the development in both hardware and software areas processing and memory units have become both faster and smaller which makes designing and developing different smart home devices more viable for building and selling commercially. This ensures that this smart home systems market is not dominated by few companies and gives a chance for anyone to try them out price wise. The second key point is the versatility of different smart home devices that are out in the market, ranging from lights to home automation and security. This variability lets people start out with few cheaper products like smart lighting or media devices and see if this is something for them.

Smart home appliances are devices that could be a common sight at many households like lights, speakers, TVs, air conditioners and so on but what makes them different is the built-in functionality for connecting to internet and then be monitored and controlled remotely. This ability to be remotely controlled and monitored makes it possible to develop automations that could further enhance the way these devices are used.

The aim of this thesis is to create smart home system that could monitor and control smart home appliances using IoT. To better demonstrate the IoT capabilities in the smart home system, a user tracking system will be developed. This system will collect user's location data in real-time, which will be used to control devices around the user. This thesis uses an open-source home automation platform and various single-board computers to handle smart home devices and track users. As the tracking should differentiate between different users, users are located via their smartphones using either WIFI or Bluetooth. Communication between the home automation platform and user tracking devices is done by using MQTT messaging protocol.

Keywords:

openHAB, openHABian, IoT, Raspberry Pi, SSH, MQTT, Python, Home Assistant, Apple HomeKit

CERCS: P170 Computer science, numerical analysis, systems, control

Targa kodu seadmete seire ja juhtimine kasutades asjade interneti seadmeid

Lühikokkuvõte:

Erinevate nutikate kodumasinate ja -süsteemide kasutamine on muutumas paljudes majapidamistes üha populaarsemaks. On mitmeid faktoreid, miks nende kasutusele võtjate arv kasvab. Mõne jaoks on see hind, sest need seadmed pole enam turul uued, seega ka enamikel seadmetel on hinnad palju käepärasemaks muutunud. Samuti on nii riist- kui ka tarkvara arengu tõttu protsessorid ja mäluüksused muutunud kiiremaks kui ka väiksemaks, mis muudab erinevate nutikodu seadmete kujundamise ja arendamise elujõulisemaks äride jaoks See tagab, et vähesed suuremad ettevõtted ei domineeri nutikodu süsteemide turgu, andes võimaluse ka väiksematel ettevõtetel oma lahendusi luua ja müüa, mis annab kõigile võimaluse neid seadmeid hinnatarkalt proovida.

Teine faktor on mitmesuguste turul olevate nutikate koduseadmete mitmekülgsus, alates tuledest kuni koduautomaatika ja turvalisuseni välja. See varieeruvus võimaldab inimestel alustada mõne odavama tootega, näiteks nutivalgustuse või meediumiseadmetega, ja vaadata, kas see on midagi nende jaoks.

Nutikad kodumasinad on seadmed, mis võivad olla paljudes majapidamistes tavalised nähtused, näiteks valgustid, kõlarid, telerid, konditsioneerid ja nii edasi, kuid mis muudab need erinevaks on sisseehitatud funktsionaalsus interneti-ühenduse loomiseks ning seejärel lasta kasutajal neid kaugelt juhtida ja seirata. See kaugjuhtimise ja seiramise võimalus võimaldab välja töötada automaatika, mis võiks veelgi täiendada nende seadmete kasutamist.

Selle lõputöö eesmärk on luua aruka kodu süsteem, mis saaks IoT abil nutikaid kodumasinaid jälgida ja juhtida. Targa kodu süsteemi IoT võimaluste paremaks demonstreerimiseks töötatakse välja kasutaja jälgimissüsteem. See süsteem kogub reaalajas kasutaja asukohateavet, mida kasutatakse kasutaja ümbritsevate seadmete juhtimiseks. Selles lõputöös kasutatakse nutikodu seadmete käitlemiseks ja kasutajate jälgimiseks avatud lähtekoodiga koduautomaatika platvormi ja mitmeid üheplaadilisi arvuteid. Kuna jälgimisel tuleks eristada erinevate kasutajate vahel, siis leitakse kasutajad nende nutitelefonide kaudu, kas WIFI või Bluetoothi abil. Koduautomaatika platvormi ja kasutajate jälgimisseadmete vahel toimub suhtlus MQTT sõnumside protokolli abil.

Märksõnad:

openHAB, openHABian, IoT, Raspberry Pi, SSH, MQTT, Python, Home Assistant, Apple HomeKit

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1. Introduction

With the progress in various information technology fields, many aspects of human society have changed. Access to information has never been quicker and easier as devices capable of accessing information anywhere with internet coverage have become very common. Today many devices around us can receive, store, process and transfer data to connected devices. This system is known as the internet of things (IoT). Over the years, the popularity in this field has been increasing rapidly. According to the GSM Association, it is expected that the number of IoT devices will grow to 25.1 billion by 2025 [1].

One of many popular real-world applications for IoT is in smart home ecosystems. Through IoT, it becomes possible to monitor, control and automate smart home devices at some level, even for private consumers with no prior knowledge in this field. A large part of the popularity also stems from the fact that it is possible to create smart home systems with relatively more nominal cost than in the past. This is due to the development and the broader availability of older technologies. Adroid Market Research, a global market research firm, points out that smart home system's take-up is growing and that smart home systems' global market is expected to exceed USD 95 billion by 2025 [2].

A smart home is essentially a home where through wireless technologies, various appliances and devices can be monitored and controlled using mobiles or other networked devices. Data generated by sensors and smart home appliances make it possible to monitor conditions at home and its devices. Devices themselves can communicate with each other or with smart home central platform through WIFI and Bluetooth. This enables for smart home solutions where through IoT capabilities, more complex automations are possible.

Smart homes are not only more popular due to the capability and affordability of smart devices but also since there are open-source smart home platforms. Some of these platforms are easy to implement and do not require huge investment. They can enable the user to tie in existing smart systems in the house to one central system. These platforms work by using wireless technologies and IoT capabilities to communicate with smart solutions from different ecosystems.

The flexibility of some open-source smart home platforms is not only limited to connecting different smart systems in homes. With some research and time, users can try to build their own solutions on top of these platforms or implement advanced automations. This level of flexibility gives the users a certain degree of control of how their smart home systems handles information flow around the house. This enables users to have better control over their privacy and be sure that their private information will not reach any third party.

The degree of control over the private information gives certain open-source smart home platforms advantage over commercial smart home platforms. This is not the only advantage of choosing open-source platform for smart home. Open-source solutions often have better transparency than commercial solutions. This means that users could have better idea how open-source smart home platform works and how it works behind the scenes, thus possibly gaining better trust than their commercial counterparts.

Although smart home platforms importance in smart home is not low, it would be almost useless without any devices to control, monitor or automate. These devices can almost be anything in the house that could gather information, use information to accomplish some task or perform simple actions when triggered. These devices could be smart entertainment, climate, kitchen, security devices and the list does not end there.

With microcontrollers and single-board computers, even normal devices could act as smart devices. As an example, it is possible to monitor ordinary devices and appliances' energy usage and implement a switch for turning them off and on. This allows for creating a home system with efficient electricity usage and thus lowers the overall cost in electricity. Due to that, homeowners could invest more into field of smart homes without having to worry about accumulating huge electricity bills.

1.1 Aim of the thesis

The aim of this thesis is to create smart home system that could monitor and control smart home appliances using IoT. To better demonstrate the IoT capabilities in the smart home system, a user tracking system will be developed. This system will collect user's location data in real-time, which will be used to control devices around the user. This thesis uses an open-source home automation platform and various single-board computers to handle smart home devices and track users. As the tracking should differentiate between different users, users are located via their smartphones using either WIFI or Bluetooth. Communication between the home automation platform and user tracking devices is done by using MQTT messaging protocol.

1.2 Outline of the thesis

This thesis consists of three stages that are:

- Research and idea stage. In this stage the outline of the system will be thought out. Following that will be a research period into different technologies that would allow to construct the envisioned system.
- Setting up home automation platform. In this stage an open-source platform OpenHAB is used for monitoring, controlling and automating various smart home devices across the house. As such this stage will consist of exploring OpenHAB functionalities and possibilities, setting up the OpenHAB, connecting devices into it and implementing automation and UI for controlling and monitoring smart home devices. This stage will also describe any problems or findings found.
- Developing user tracking system. This stage will cover how house user tracking system was developed. This will cover the research into different approaches and what approach was used in the end. As such there will also be steps taken in development phase and all risen problems will also be covered.

As for thesis structure it is as following:

- Background, section for information about the technologies and tools used in this thesis.

- Requirements, section describing use cases and requirements for the system developed in this thesis.
- System architecture, section that will cover the following subsections:
  - House automation system, section about setting up home automation platform, connecting smart home devices into it and automating said devices.
  - User tracking system, section about developing system for tracking users in the house.
- Implementation Challenges, section describing different challenges that were faced when developing the system.
- Conclusion, section describing the work accomplished.
- Appendix, section with the following:
  - Appendix I, section that showcases the automation script.
  - Appendix II, section that showcases the user tracking script.
  - Appendix III, section for license.

2. Background

This section will give information about the system built in the thesis and introduce different components and technologies used in the implementation. This section will cover different open-source house automation platforms and explain why openHAB was used in the end. Next, there will be information about what tools and hardware will be used for the user tracking system. Finally, the smart home devices that are going to be used in this thesis will be introduced.

2.1 Idea

This thesis aims to create an IoT-based smart home central system that could control, monitor, and automate different smart home devices in the house. This system will make use of a small single-board computer and open-source platform capabilities of good modularity, easy development and low cost. The user tracking system will be built with the purpose to gather data about user's whereabouts in the house. This data will be used to automate smart devices around the house to respond when users are near them or have left their range. The main purpose of interaction between user location tracking and smart appliance control is to show what IoT-based smart home system is capable of.

Not only is the location of users monitored but also the smart home devices. The smart home system will use the data gathered from each smart home device and enable a manual control panel for house users to access. Said control panel would show each device's state, have an option to control these devices manually, and possibly show any data that is useful for the user, for example, time, date, the climate in and out of the house, energy consumption and more. This control panel will be able in both web and mobile application form.

An example of a possible scenario that should be possible with this system: Joe arrives at home. The system detects that and will turn on lights in the vestibule and cloakroom. Next, as it is a warm summer day, the system will turn the air conditioner on. As Joe reaches his room, the system will turn off lights in previous locations and turn on the Joes room lights.

After some time, Joe leaves, and the system will turn off all the active devices and as indoor temperature is still relatively high, the air conditioner is set to run in an eco-mode for more efficient power consumption.

Later, Joes wife reaches home and starts doing laundry. As she is working in the laundry room and going around the house collecting old bedsheets, the system automatically switching on and off lights in the rooms she is visiting.

According to this example, the system should work with multiple users and should be able to differentiate between them. This will also make it possible for customised automations. The following subsections will introduce the components and tools used for making this system possible.

2.2 Home Automation Platform

A home automation platform is a system that is responsible for monitoring and controlling home devices like lightning, climate, entertainment systems, appliances and even home security. These devices are typically connected to a central hub that is managed by a house automation platform. There are many competing vendors for these platforms, and there are even several open-source systems.

2.2.1 Platforms

For this thesis, the home automation platform must be able to handle multiple devices efficiently and have support for MQTT for simple and effective communication between devices and includes support for a large variety of smart home devices and appliances. These platforms should also have well-prepared documentation for understanding platforms' capabilities and limits and have an active user base and community for quickly getting answers to any specific question.

One other criterion for the home automation platform is to be IoT friendly and be efficient in its use of resources. As this smart home system will use a small single-board computer, the preferred platform would need to be optimised to run on it. Stability is also a significant factor, and there should not be any problem with storing and keeping safe the gathered data for a certain amount of time.

The following subsections will introduce few platforms that potentially cover the set criteria and could be used to build similar systems created in this thesis. Based on these platforms, the one used in this thesis will be selected and further presented and discussed. One commercial platform is also introduced for better comparison.

2.2.1.1 Home Assistant

Home automation software Home Assistant[1] is a free and open-source platform for creating central control systems for smart home devices and appliances. This platform is designed around and written in Python, enabling Python's use in further custom development. Since November 2020, according to their official integration page [3], Home Assistant has over 1700 modular add-ons that will allow it to use various smart home devices, services, and systems.

Project Home Assistant started in September of 2013, and after few months in November, the core project was first published publicly on GitHub. The founder of this project is Paulus Schoutsen, and according to him, this project's goal was to be the platform for the home. Since the initial release, the project was more of a hobby project aimed at controlling Philips Hue lights [4]. Now, the Home Assistant community has risen in numbers, and in 2020 GitHub listed Home Assistant as second place in Python packages with the most active contributors.

As mentioned by the founder of Home Assistant in one article [4], this project aimed to create smart home platform that could control everything from a central point. His idea was to make smart home platform that focuses on smart home systems' usability and adaptivity. In his vision, that he talked about in one of his blog posts [5], the home automation system should not be something that would become cumbersome to use but instead be something that all people in the household would find helpful. This means that system should work

[1] https://www.home-assistant.io/

flawlessly, blend with everyday workflow and should run at home. Overall, according to the blog post, the vision for Home Automation was to be able to develop smart home systems that would never get in the way or annoy but would be missed when not working.

Home Assistant can be used in numerous ways. Recommended way would be to use their operating system in a dedicated system, for example, on Raspberry Pi. It is also available as a container-based solution that, for example, could run on Docker. After initial configuration, the platform could already detect some smart home appliances on the home network. Other devices that were not automatically found can easily be added through UI. if there are integrations available to those devices.

According to Home Assistant documentation page [6], automation in Home Assistant can be accomplished in multiple ways. The easiest way to implement automation is through UI using the Automation Editor feature. This builds automation that works by waiting for a trigger, then checking conditions and finally calling an action. For more advanced automating, Home Assistant has Templating feature that uses Jinja2 templating engine and syntax. Using this, users could benefit from a wider variety of operations and custom variables to build more in-depth automations and system rules. Home Assistant is also capable of running python scripts. This means that users are not limited to use only the provided templating engine but could push their imagination to the limit and implement functionalities that are only limited by Python and hardware capabilities.

2.2.1.2 openHAB

The open Home Automation Bus, or openHAB[2] for short, is an open-source home automation platform designed to be a central piece for a smart home [7]. openHAB is entirely written in Java and is based on the Eclipse SmartHome framework. It is also very modular, and its base platform can be enhanced and extended through different add-ons. This allows for openHAB to be able to use different kinds of services or communicate with various home automation solutions and devices. The openHAB supports close to 3000 various add-ons for both different services and devices as of writing this thesis.

In Kai Kreuzer blog post [8] he briefly explains about the origin of openHAB and the process of development of both openHAB and the team behind it. According to him, the openHAB project was initially released in 2010 by Kai Kreuzer and has since then become very popular. In 2013 openHAB core framework was contributed to Eclipse Foundation and became the Eclipse SmartHome project. According to him, this move allowed openHAB to truly become an open-source project with well protected and rigid intellectual property management. This enabled other companies and developers to use the openHAB core in their own solutions and ultimately helped with the openHAB community's growth and contributions. Based on the Eclipse SmartHome framework, new development branches of openHAB were created as versions 2.X and 3.X.

On openHAB vison page [9], the idea behind openHAB was to combat the situation where the user comes up with ideas and wishes on how to use different devices and systems but

[2] https://www.openhab.org/

that were not supported out of the box or intended in their use cases. Thus, according to them, this project was imagined as a central integration point between all of the different systems and services. As the vision behind this project was to consider users' wishes, handling information and privacy of users was also one of the key points. Considering that, openHAB gives their user the option to decide how they wish to control the data movement in and out of the local system.

According to the team behind the openHAB [9], the openHAB was never intended to replace the existing solutions but rather enhance them, it is considered a system of systems. This means that all the sub-sub systems are to be configured and set up independently, leaving openHAB to only focus on these sub-systems' daily use. These sub-systems can further be broken down into items that are used in defining automation rules and UIs of the system. In openHAB, the notion of an item is a data-centred functional building block, and it does not matter whether this item is a device or some web service, which makes the concept of item abstract. They say that, this concept makes it easy to switch between different services and devices without changing the automation rules and UIs defined beforehand.

openHAB automations are handled through a lightweight rule engine as such automations are referred to as rules in openHAB. These rules work using a trigger-based action activation model. These rules can be defined few different ways. One of the ways is to use the UI based simple rule creator. Their users can easily choose triggers and corresponding actions to be executed.

According to openHAB documentation [10], for more in-depth automation, users can write their own rule files. Documentation mentions that, as the rule syntax is based on Xbase and is similar to Xtend, users can refer to Xtend documentation when writing rules. Also, the process of writing rules is made easier through openHAB VS Code Extension, which offers syntax checks, colouring, and many more valuable features. The rule engine can also use JavaScript scripts which gives the users even more freedom when creating automations.

2.2.1.3 Apple HomeKit

HomeKit[3] is according to Apple's developer page [11] a framework for smart home devices and appliances that are HomeKit enabled by manufactures. This software framework only works with applications that run on Apples operating system. These applications like home allow users to configure, monitor, control and automate their smart devices.

Apple HomeKit was first released with iOS 8 on 2014. At release the framework worked with third-party applications which enabled them to have interface for HomeKit devices. On 2016 Apple released their own official application called Home for managing all HomeKit enabled devices and made simple automations possible through that. Home application only worked with devices running iOS 10 and watchOS 3 but later, on 2018, this application was also released Apples computers, running macOS 10.14.

According to Apple [12], the main idea behind HomeKit was to simplify the tasks around the home. For that HomeKit was developed to act as a set of tools, enabling Apple and other

[3] https://developer.apple.com/documentation/homekit#//apple_ref/doc/uid/TP40015050

people to build solutions for their needs. These solutions could allow for high level management over accessories and devices.

Although HomeKit could be used by users to develop their own complex smart home solution for monitoring, controlling or automating HomeKit enabled devices, it does have a minus over open-source platforms. That minus is that the number of supported devices is little. According to J. Porter's article [13], as of 2019 only 450 devices were marked as compatible. This may be due to MFI program, that manufactures must enrol in, in order to have permission to add HomeKit capabilities to their devices. This means that only certain manufactures can develop devices that work with HomeKit.

2.2.2 Chosen Platform

The requirements and capabilities expected from the home automation platform used in this thesis were numerus. From three solutions only openHAB and Home Assistant met those requirements. Both are essentially very similar in their capabilities with no noticeable limitation for neither of them. However, openHAB was picked to be the platform to be used in this thesis.

One of the key differences between the two platforms, that decided which platform was going to be used in this thesis, was documentation quality. The documentation for automation syntax and different add-ons were more in-depth and extensive in openHAB. Both platforms have also included examples of possible use-cases for better understanding, but openHAB seemed to have more comprehensive and overall, more examples than Home Assistant.

As for other requirements, openHAB is configured to run on a myriad of different platforms such as Windows, macOS and Linux and even in container-based environments like Docker. It will work with almost any hardware in the range of limited single-board computers to server computers. According to their documentation, the system will behave well with 2 GB of random-access memory and 16 GB of low-speed storage using SD card. This means that Raspberry Pi's 2 to 4 are good enough, where Pi 4 would give the best experience.

Due to openHAB's modular design with close to 3000 add-ons and well-established support for MQTT, it is fair to say that the openHAB platform is very IoT friendly. Their add-ons enable using a myriad of different IoT devices and services, which lets the users design solutions for practically any IoT device. The well thought out MQTT support makes it possible for other IoT systems and devices to communicate effortlessly with openHAB.

2.3 User tracking system

This thesis aims to show what IoT can accomplish in a smart home environment. This thesis will develop a user tracking system that could gather data about user's whereabouts around the house and transmit that data through MQTT to the home automation platform. The user tracking system will essentially allow for more complex automations and smart home system control thanks to IoT capabilities.

User tracking systems can be implemented in a variety of different forms. One way to divide them would be a system that either could or could not differentiate between users. Both systems have their use-cases, methodology and implementations. For a system that could not acknowledge one user from another, it is usually implemented to sense movement or existence of entity around the system. For this, these systems would use sensors that could detect movement or heat emission from a foreign entity. An example of where these systems are most often found is in security to detect intrusion.

The tracking systems that could differentiate between users in its detection range are usually implemented to keep track of users' whereabouts. This is done for a couple of reasons. One of them is for security. If it is possible to know where somebody is, then it is possible to seal off access to areas they are not privileged for. Many offices use this approach to keep the personnel to their specified areas.

The second use case is for controlling and automating the environment around the user. This is mainly implemented in smart home solutions for more complex, personalised and convenient automations. An example of a home automation system with this tracking system would be when one user could enter the bathroom and bath would be preheated to their preferred temperature.

For this system to differentiate between users and track their whereabouts, the system would need to read identification data from the user when they enter a new area. The data used for identification could be always on the user, for example, biometrics data or an external data carrier. In the case of using biometrics, then the system could either implement face recognition through cameras or fingerprint scanners at each entrance.

Using external identity data carriers, the system could be implemented in various ways depending on the carrier. The carriers could share the data two ways, either in contact or near range. For contact range, the carriers are not active and only passively share data once they come in contact with some sort of reader. This system could be implemented through the usage of personalised key cards and card readers at each entrance. As for the near range carriers, they could be active and connect to nearby scanners to transmit identification data from range. This means that users would not need to take action at each entrance but simply pass by, and the system would record their whereabouts by itself.

For this thesis, the tracking system will be able to differentiate between users using the external identity data carriers that the system from a close range could read. This will make it possible to implement automations near the users seamlessly and be more convenient as the user does not need to perform an identification action at each entrance. The system will consist of multiple trackers, one per room, to automatically detect the user when they are near or in the room.

2.3.1 Trackers hardware

This thesis's user tracking system will be a series of stand-alone devices capable of wireless technologies like WIFI and Bluetooth. These devices are going to be divided around the house, one device per room. WIFI will be required to send data from each device to the home automation platform.

Wireless technology will also be used to track users. In order to discover users, tracking devices in different rooms will be using Bluetooth to find user carried Bluetooth devices.

By doing so, if a user carried Bluetooth device is discovered, for example, their phone, the tracker can check the signal strength from it to the device. After one iteration of scanning, the device will send data consisting of discovered devices and their signal strengths to it via MQTT.

In order to realise this idea, the stand-alone devices need to be programmable and capable of running continuously. There are multiple different hardware options that could be considered. Two main options are to use either Arduino single-board microcontrollers or Raspberry Pi single-board computers. Both options have their strengths and weaknesses.

Arduino is an open-source company that designs and manufactures single-board microcontrollers, according to Arduinos introduction page [14]. From a wide variety of controllers that they manufacture, some can use WIFI and Bluetooth. In essence, Arduino microcontrollers are perfect for interacting with sensors and handling data in small quantities. Depending on the use case, they can be very affordable. The main downside of Arduino microcontrollers is that they are capable of primarily small singular tasks, and some controllers are not easy to source.

Raspberry Pis are developed by Raspberry Pi Foundation in the United Kingdoms. The Raspberry Pi is a single-board computer that can run operating systems and graphical output. Their use-cases are broader compared to Arduino microcontrollers as they are not that limited in hardware and are also affordable.

The tracking system stand-alone devices in this thesis make use of Raspberry Pi Zero W single-board computers. Although Arduino also has microcontrollers that would fit into the tracking system use-case, Raspberry Pi Zero Ws were more accessible. Also, these Raspberry Pis enabled more tinkering and researching into different ideas of discovering user.

2.3.2 Trackers software

For trackers to work, they needed to have software capable of utilising the hardware. As the trackers are running Linux based Raspberry Pi OS Lite operating system, then there were multiple different ways to implement user tracking and discovery. One of the ways was to use some familiar programming language and its modules to implement the Bluetooth discovering and data transfer over MQTT. The other option was to find an already existing open-source solution and develop software on top of that.

After searching for possible open-source solutions, there was only one that suited the use case called reelyActive. reelyActive was founded in 2012 with the idea to create a cloud-based active RFID system. Over the years, their team and project have grown, and as of 2017, the solution is capable of running on Raspberry Pi and supports identifying Bluetooth devices.

reelyActive seemed to be an excellent open-source solution that could be used in implementing the software for trackers. But due to aged and almost non-existent documentation and support for seemingly only Bluetooth Low Energy technology, this open-source solution was not used.

As there were no suitable open-source solutions to implement trackers software, the only way was to implement a solution from scratch. The user tracking system software will be implemented using Python, as it has support for Bluetooth and MQTT.

2.4 Smart Home Devices

In this thesis, due to limited amount of resources, Philips Hue smart light bulbs will be used as smart home devices. In total 3 Philips Hue ambient and colour smart lights will be used.

The Philips Hue line of smart LED lights is manufactured by Signify N.V. This line of LED lamps can have colour changing, light level, and temperature control capabilities and come in various forms. Apart from more traditional bulbs, they can also be in light strip, outdoor lamp, floodlamp, and many other forms.

What makes Philips Hue lights smart is the capability for them to be controlled wirelessly as specified on Philips Hue about page [15]. For that, there are few ways. With newer lights, it is possible to connect to them over Bluetooth and manage them through smartphone applications or computers. The second way is to have Hue Bridge, a central controller for Philips Hue lights. The Hue Bridge can be connected to home internet through an ethernet cable, and then Hue Bridge will handle each light by itself. This means that users do not need to connect to each light separately and easily control them through the internet.

This thesis uses only these lights as they are capable of different automations and thus sufficient for demonstrating IoT capabilities in monitoring and controlling smart home devices.

3. Requirements

This section will give overview of functional and non-functional requirements. These requirements will describe what was focused on in setup and development process of automation and user tracking systems. Described requirements are based on presented idea in section 2 and goals set with thesis supervisor.

3.1 Functional requirements

This subsection will describe the functionalities that both smart home and tracking system will provide to users. Description will follow the schema of functionality and then description of functionality.

- The smart home system should show data about connected devices. This means that users should be able to see any meaningful information about the devices connected to the smart home system through use of user interface. For example, the data could be about the power state of devices or information collected by sensors.
- User should be able to manually control the devices. The devices that are connected to smart home system and can be controlled should be controllable by user. This means that user should be able to use user interface to control devices in any meaningful way.
- User should be able to add new devices to the smart home system easily. This means that smart home system should have functionality to easily add any supported device. For example, this system could find new devices by looking through devices connected to network.
- The smart home system should allow writing automation scripts and rules. This means that smart home system used in this thesis should allow for writing custom automation scripts for devices connected to it.
- The smart home system should be able to store data to it. This means that the smart home system should enable for automatically or manually storing information to it or to the cloud. This is for saving states of devices for automation and monitor purposes and creating graphs using selected data collected over time.
- Guests locations should be included in location-based automations in the smart home system. This means that smart home system could make use of information regarding guests' location and enable generic automations based on their location.
- The user tracking system should be able to frequently share information with the smart home system. This means that the smart home system should have access to user location that is updated frequently. This enables for user location-based automations that could trigger whenever user is near to a certain room or location.
- The user tracking system trackers should be plugged in and out without any problem. This means that users could unplug the trackers from the electricity and re-plug them without facing any problem. The trackers should automatically start scanning for users when turned on.

- The user tracking system should be able to track guests' locations. This means that user tracking systems trackers would not only track the users but also any unknown user and their devices. This would enable for the smart home system automations to work with guest locations as well.

3.2 Non-functional requirements

This subsection will describe the non-functional requirements that act as a criterion for both the smart home system and the user tracking system. Descriptions will follow the schema of non-functionality and then description of non-functionality.

- The smart home system user location automation should trigger almost immediately when user's location changes. This means that there should not be large time difference between user changing their location and location-based automations triggering.
- The smart home system should allow for multiple automations to trigger simultaneously. This means that system could handle multiple automations at the same time which would prevent tasks from piling up.

4. Smart Home System Architecture

This section will cover the architecture and configuration of the home automation platform and the user tracking system. The following subsections will describe how both systems were designed and cover the steps taken to build them.

4.1 System Overview

The home automation system built in this thesis consists of two sub-systems: the *home automation platform* and the *user tracking system*. The *home automation platform* will be the central system that handles the smart devices and services, enables automations and user interaction. The *user tracking system* consists of stand-alone tracker devices, each responsible for specific rooms in the home. This tracking system could be viewed as a service for the home automation platform. Essentially the tracking system is responsible for collecting user location data and sending that to the home automation platform for user location-based automations.

These two sub-systems communicate through the MQTT network. The network will consist of an MQTT broker, hosted by the home automation platform, and MQTT clients on each tracker. Each MQTT client on trackers will be publishing collected data to the MQTT broker. The home automation platform will also have an MQTT client that is subscribed to the broker. This enables the platform to manage sent data and use that information in automation. The described system communication network is represented in Figure 1.

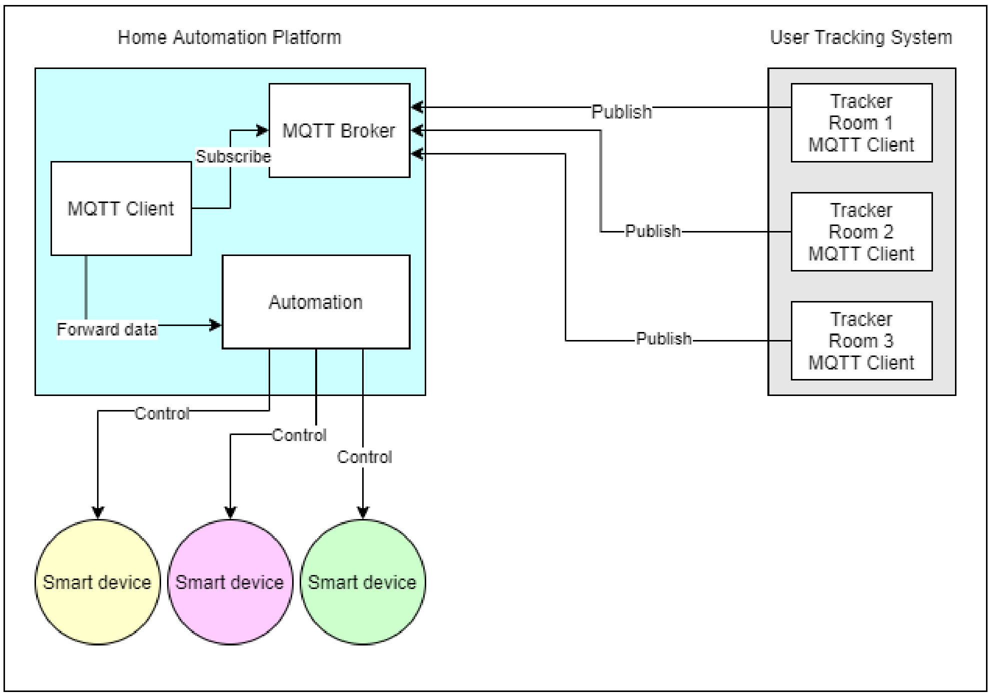


Figure 1. Communication network between systems.

4.2 Home Automation Platform

The following subsections will cover the overview of the system, the setup and configuration processes, adding the devices and services to the platform and implementing the automation.

4.2.1 System

The home automation platform will consist of two parts, the software and hardware. The software sets the requirements for hardware. In this thesis, the software used is the open-source home automation platform openHAB. Based on openHAB documentation, the platform is suitable to run on various systems and hardware. One of the hardware that is supported by the openHAB is the Raspberry Pi line of single-board computers.

This thesis uses Raspberry Pi 4 for containing and running the openHAB platform. Raspberry Pi 4 has different available configurations on the market, and one used in the thesis has the following hardware specifications:

- CPU – Broadcom BCM2711, Quad-core Cortex-A72, 64-bit, 1.5 GHz
- RAM – 4GB LPDDR4-3200 SDRAM
- WIFI – 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless
- Bluetooth – Bluetooth 5.0 and BLE

This single-board computer does not have any built-in storage, and as such, the storage is handled through a Micro-SD card as the board has an available slot for it. There are other features on the board like graphical output ports over micro-HDMI, Gigabit Ethernet port and USB 3.0 and 2.0 ports, but these do not have any purpose in this thesis.

The Raspberry Pi 4 single-board computer will be running a Linux system setup by openHAB called openHABian[4]. openHABian is based on standard Raspberry Pi OS Lite and designed as a headless system. It has plenty of features that make setting up openHAB easy. One of the features is that openHABian will set up all the necessary tools on the first boot and install the latest packages on the first boot. This makes it easy for new users that have no prior experience with Linux based systems as they only need to provide ethernet access to Raspberry Pi and leave it to set itself up.

Other than that, openHABian also comes with Linux packages and pre-installed settings that enable users to set up more advanced system configurations. A few of these packages are for setting up security measures, providing IoT-friendly communication through MQTT, enabling databases and data visualisation, and providing secure remote connectivity through VPN. All of these packages and more can be enabled and configured through the openHABian Configuration Tool. The user could also configure the system setting to create backups and update openHAB and openHABian through this tool.

4.2.2 Initial Setup

[4] https://www.openhab.org/docs/installation/openhabian.html

With hardware and software introduced in the previous section, the home automation platform can be set up.

The home automation platform setup on Raspberry Pi single-board computer requires the following:

- Raspberry Pi 4
- openHABian image file
- Flash tool
- 16 GB Micro-SD card
- Micro-SD card reader
- Ethernet access

To get the openHAB platform running on the Raspberry Pi 4, the openHABian image file is needed. It can be acquired on the openHAB project GitHub page[5], accessed easily through the documentation page, covering the openHABian installation process. There are multiple versions of openHABian images, but the image used in this thesis is version 1.6.1, which has version 2 of openHAB called openHAB 2. This image was at the time of installation the newest available stable release. Since then, there have been new releases with updates to openHAB tools and packages. Even a new version of openHAB has been introduced, which is openHAB 3.

After the image file has been acquired, it can be flashed onto a Micro-SD card. There are plenty of tools available for flashing, but one used in this thesis is balenaEtcher[6] by balena. This tool is relatively easy to use, as it only needs few inputs from the user. Using this tool first image file was selected and then the destination. It should be noted that when using balenaEtcher, the openHABian image file should be unpackaged before flashing to a storage device.

Another requirement for flashing image on a Micro-SD card is to have a Micro-SD card with 16 GB of storage and a way for the computer to manage the flashing process. The 16 GB of storage is not mandatory for the Micro-SD card as the image does not take up a large amount of space but is recommended by the documentation. For computer to access Micro-SD card, it needs to have a SD card reader compatible with Micro-SD cards. As the computer used for flashing purpose already had an inbuilt reader, no additional tool was necessary.

After the flashing process has finished, the next step is to set up openHAB on Raspberry Pi 4. The Micro-SD card with flashed image needs to be installed on the Raspberry Pi 4, connected to the internet either through WIFI or Ethernet cable. Using the Ethernet cable is easier as connecting with WIFI involves modifying the configuration file with WIFI SSID and password before the first boot. After the Micro-SD card is installed and a way for connecting to the internet is provided, the single-board computer can be booted.

[5] https://github.com/openhab/openhabian
[6] https://www.balena.io/etcher/

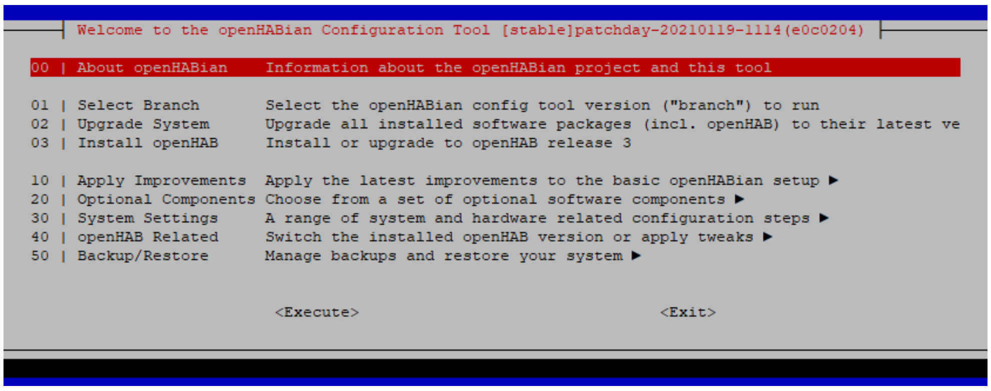
After the Raspberry Pi 4 is booted for the first time, it will automatically set everything up. This process length is entirely based on the internet connection, Micro-SD card writes and reads speed and the Raspberry Pi computers processing capabilities. In the installation documentation, it is mentioned that this process could take around 15 to 45 minutes. It should be noted that Raspberry Pi 4 in this thesis took about 20 minutes to finish this process.

When the process finishes, the openHAB web server is set up. Web server's user interface can be accessed on any computer on network through a web browser with the network router designated IP address for Raspberry Pi 4 on port 8080. For example, when the router sets the IP address for the Raspberry device to be 192.168.1.3, then the user interface can be accessed on a web browser at 192.168.1.3:8080.

4.2.3 Configuration

This section will focus on configuring the home automation platform openHAB. Through this process, connecting various devices and services to openHAB is possible. The user interface will be implemented and configured to enable the user to monitor and manually control added devices and services. This section will also describe the automation process and how user location data will be used.

There are a couple of ways to configure openHAB depending on what the user is trying to achieve. Suppose the user aims to configure the openHAB system like settings concerning updating, restoring from backups and creating backups or changing hardware features. In that case, openHABian has a tool called Configuration Tool. This can be accessed through connection to the command line console running on the openHABian image. The described tool is represented in Figure 2.



```
Welcome to the openHABian Configuration Tool [stable]patchday-20210119-1114(e0c0204)

00 | About openHABian     Information about the openHABian project and this tool

01 | Select Branch        Select the openHABian config tool version ("branch") to run
02 | Upgrade System       Upgrade all installed software packages (incl. openHAB) to their latest ve
03 | Install openHAB      Install or upgrade to openHAB release 3

10 | Apply Improvements   Apply the latest improvements to the basic openHABian setup ▶
20 | Optional Components Choose from a set of optional software components ▶
30 | System Settings      A range of system and hardware related configuration steps ▶
40 | openHAB Related      Switch the installed openHAB version or apply tweaks ▶
50 | Backup/Restore       Manage backups and restore your system ▶


                  <Execute>                              <Exit>
```

Figure 2. Configuration Tool.

Other than that, the configuration can be aimed to add new devices, services, packages, implement user interfaces and add automation. This can be done in two different ways. The first option is to use the openHAB web user interface. Through the Paper UI tool, users could install packages and modules for adding various devices and services and configure them. This tool also allows to monitor and control added devices and services and add automations to them. The described tool and actions are represented in Figures 3, 4 and 5.

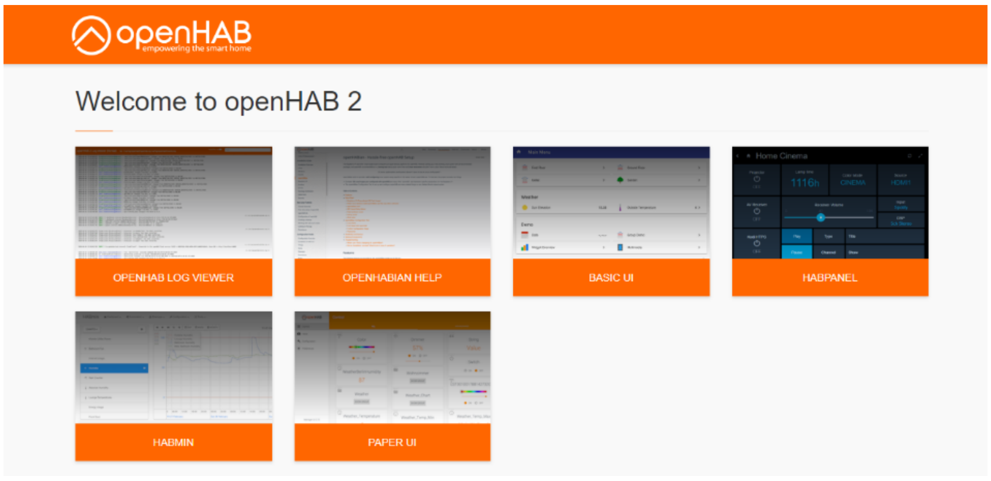
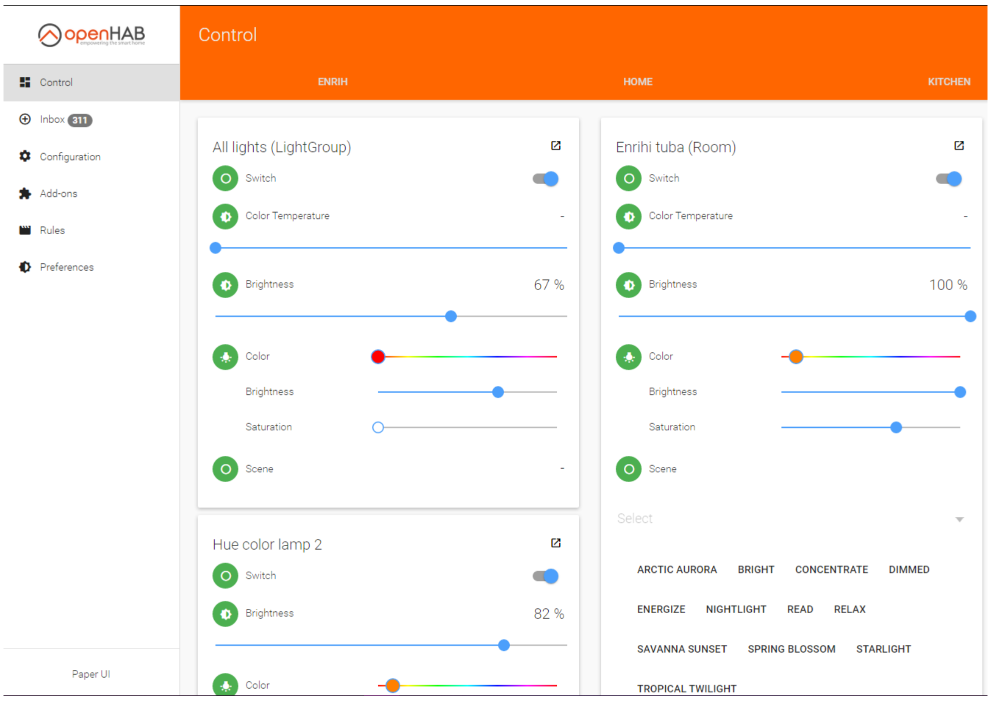


Figure 3. List of tools on openHAB.



All lights (LightGroup)

Switch

Color Temperature -

Brightness 67 %

Color

Brightness

Saturation

Scene -

Hue color lamp 2

Switch

Brightness 82 %

Color

Enrihi tuba (Room)

Switch

Color Temperature -

Brightness 100 %

Color

Brightness

Saturation

Scene

Select

ARCTIC AURORA BRIGHT CONCENTRATE DIMMED

ENERGIZE NIGHTLIGHT READ RELAX

SAVANNA SUNSET SPRING BLOSSOM STARLIGHT

TROPICAL TWILIGHT

Figure 4. Control menu view.

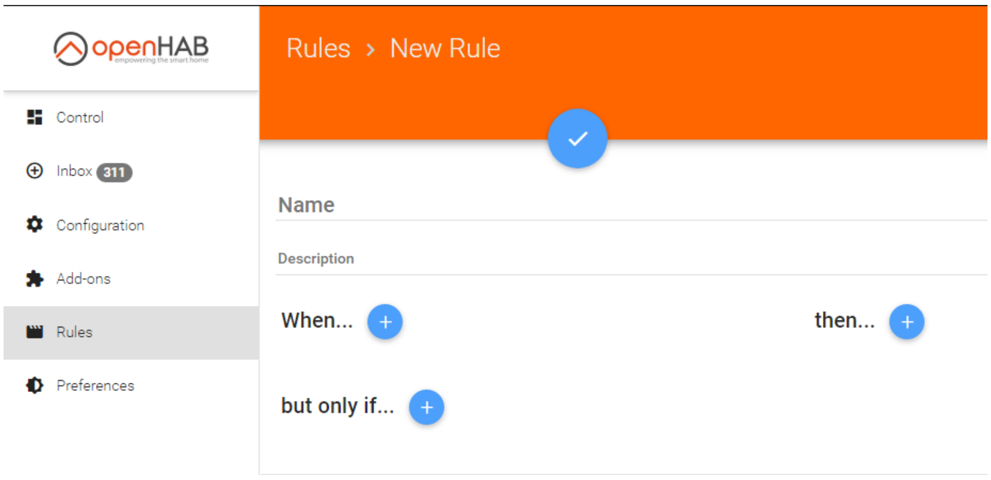


Figure 5. New rule template.

The second option is to modify configuration files on the openHAB system directly. Through this option, users could achieve the same results, that was possible with the Paper UI tool on the openHAB web user interface. This option also allows users to write more advanced automations and develop JavaScript-based scripts, making it more flexible than the first option. The downside of this option is that modifying and writing custom configuration files may be more complex than using the Paper UI tool and could require more in-depth knowledge. The upside is that this option allows for more advanced system setups and configurations.

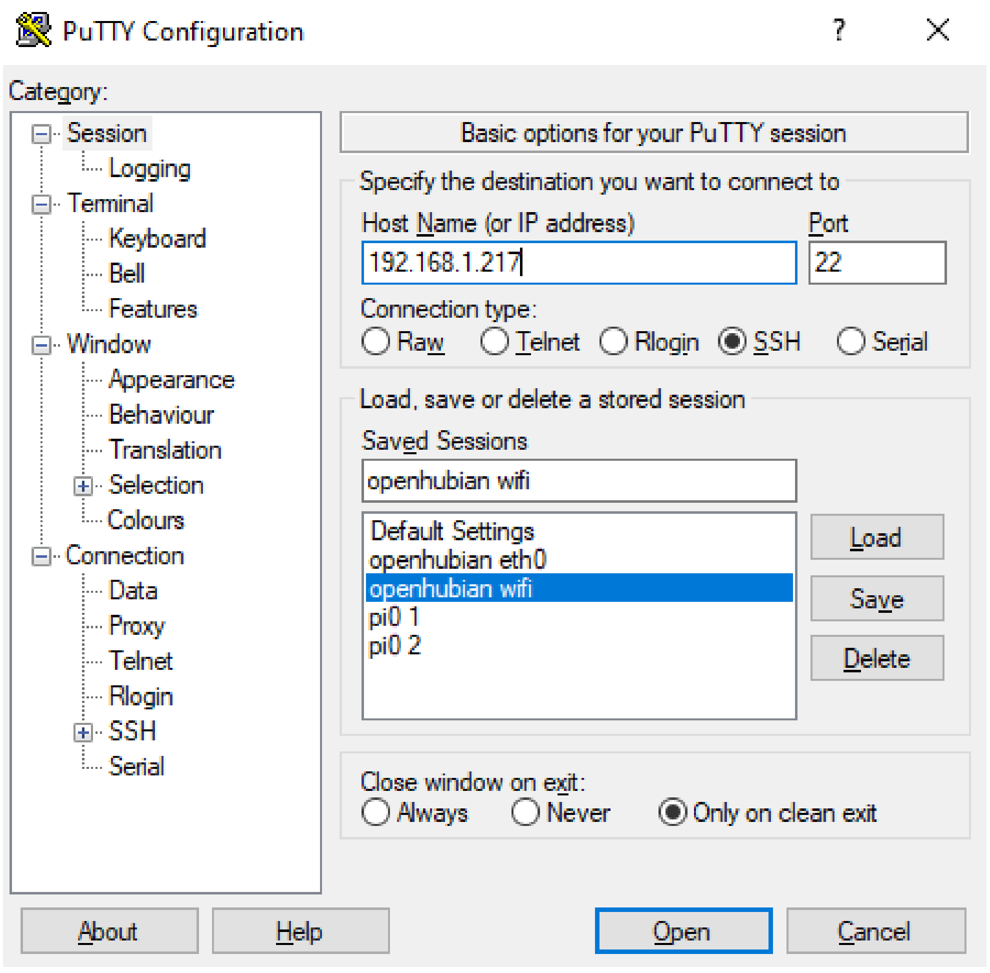
In this thesis, the first option is used to discover and add the devices, and the second option is used for installing packages and implementing user interfaces and automation. This will be further explained in the following subsections. The following subsections will also cover the process of configuring openHAB.

4.2.3.1 Configuring openHAB

After the openHABian system has booted for the first time, the openHAB platform does not need any additional configuration to start functioning. Although it is not required, some advanced features will not work without configuring the system. One of the advanced features is MQTT capability, which is by default disabled. As the system built in this thesis requires this functionality, then it should be enabled.

To configure the openHAB, the openHABian Configuration Tool was used. This tool can be accessed through the command line console on the openHABian system. To access the openHABian system, a SSH connection was established using PuTTY[7], which is shown in Figure 6. To open a PuTTY session in the openHABian system, the IP address of Raspberry Pi 4 and SSH port is required as the destination. The IP address was found using the network routers interface. As for the SSH port, it is by default 22.

[7] https://www.putty.org/


PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 192.168.1.217

Port: 22

Connection type: ( ) Raw ( ) Telnet ( ) Rlogin [✓ SSH] ( ) Serial

Load, save or delete a stored session

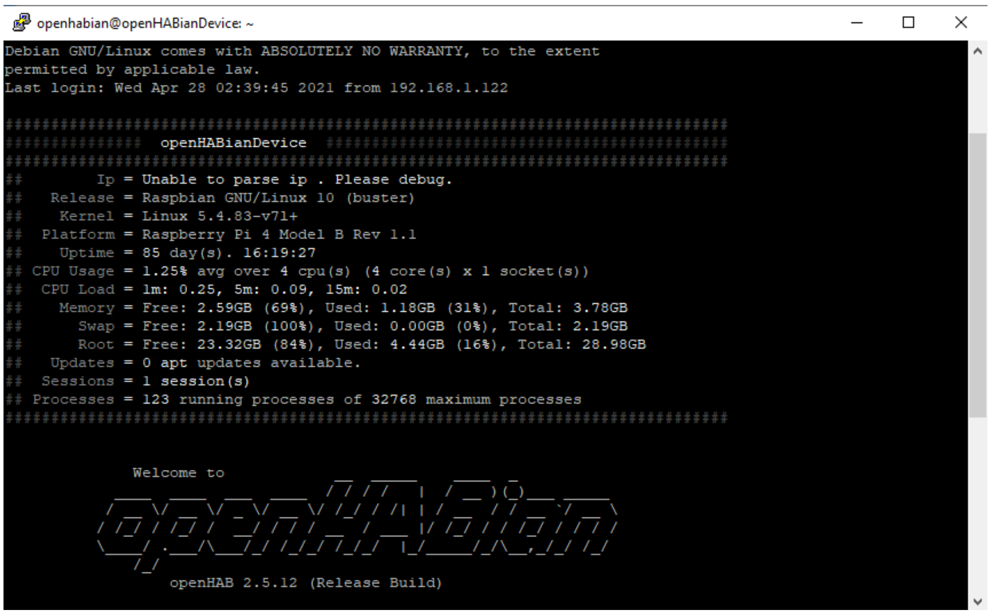
Saved Sessions: openhubian wifi

Default Settings
openhubian eth0
openhubian wifi
pi0 1
pi0 2

Close window on exit: ( ) Always ( ) Never [✓ Only on clean exit]

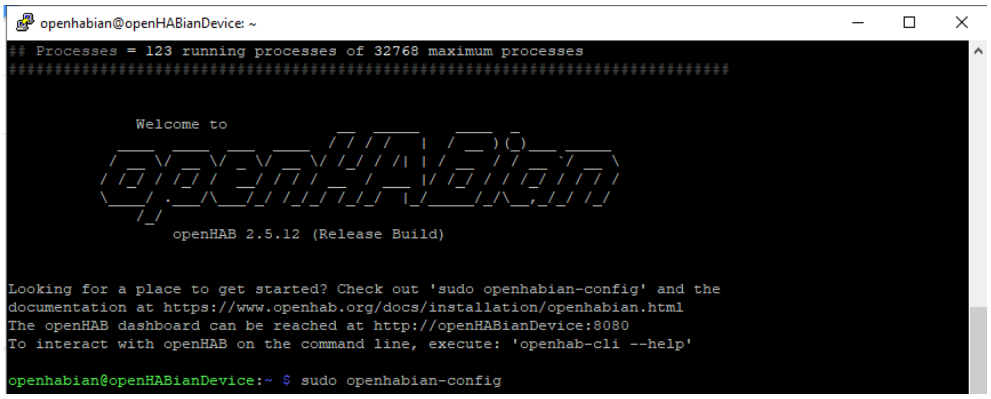
Figure 6. SSH configuration tool

After the SSH connection is opened, PuTTY will display the command line console. As the openHABian system is protected, then the console will request a username and password, which is by default “openhabian” for both. After successful login, a welcome screen is displayed, which is shown in Figure 7. As the openHABian image is based on the Linux kernel, the console can be operated with Linux commands. The configuration tool can be accessed with the command “sudo openhabian-config”, as shown in Figure 8.



```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Apr 28 02:39:45 2021 from 192.168.1.122

################################################################################
###############  openHABianDevice  #############################################
################################################################################
##        Ip = Unable to parse ip . Please debug.
##   Release = Raspbian GNU/Linux 10 (buster)
##    Kernel = Linux 5.4.83-v7l+
##  Platform = Raspberry Pi 4 Model B Rev 1.1
##    Uptime = 85 day(s). 16:19:27
## CPU Usage = 1.25% avg over 4 cpu(s) (4 core(s) x 1 socket(s))
##  CPU Load = 1m: 0.25, 5m: 0.09, 15m: 0.02
##    Memory = Free: 2.59GB (69%), Used: 1.18GB (31%), Total: 3.78GB
##      Swap = Free: 2.19GB (100%), Used: 0.00GB (0%), Total: 2.19GB
##      Root = Free: 23.32GB (84%), Used: 4.44GB (16%), Total: 28.98GB
##   Updates = 0 apt updates available.
##  Sessions = 1 session(s)
## Processes = 123 running processes of 32768 maximum processes
################################################################################


              Welcome to
                                    __  _____    ____  _
            ____  ____  ___  ____  / / / /   |  / __ )(_)___ _____
           / __ \/ __ \/ _ \/ __ \/ /_/ / /| | / __  / / __ `/ __ \
          / /_/ / /_/ /  __/ / / / __  / ___ |/ /_/ / / /_/ / / / /
          \____/ .___/\___/_/ /_/_/ /_/_/  |_/_____/_/\__,_/_/ /_/
              /_/
                  openHAB 2.5.12 (Release Build)
```

Figure 7. openHABian welcome screen.



```
## Processes = 123 running processes of 32768 maximum processes
################################################################################


              Welcome to
                                    __  _____    ____  _
            ____  ____  ___  ____  / / / /   |  / __ )(_)___ _____
           / __ \/ __ \/ _ \/ __ \/ /_/ / /| | / __  / / __ `/ __ \
          / /_/ / /_/ /  __/ / / / __  / ___ |/ /_/ / / /_/ / / / /
          \____/ .___/\___/_/ /_/_/ /_/_/  |_/_____/_/\__,_/_/ /_/
              /_/
                  openHAB 2.5.12 (Release Build)


Looking for a place to get started? Check out 'sudo openhabian-config' and the
documentation at https://www.openhab.org/docs/installation/openhabian.html
The openHAB dashboard can be reached at http://openHABianDevice:8080
To interact with openHAB on the command line, execute: 'openhab-cli --help'

openhabian@openHABianDevice:~ $ sudo openhabian-config
```

Figure 8. Command for accessing the openHABian Configuration Tool.

In the tool menu, the MQTT can be enabled under the “Optional Components” tab. The Mosquitto[8] MQTT broker is used by the system built in this thesis and can be enabled by selecting it in the menu. Additionally to MQTT, WIFI was also enabled, and that was done under the “System Settings” tab.

After that, the tool can be exited, and the SSH connection closed. It should be noted that if enabling WIFI, then the system will be assigned a new IP address. This means that to access web-based openHAB user interface or establish SSH connection, the new IP address should be used.

4.2.3.2 *Adding Devices & Services*

openHAB documentation states that each device connected to openHAB is different and needs base components to represent all of them [16]. The main base components responsible for adding and managing devices and services are the Add-ons, Things and Items. Each of these base components has its own function.

[8] https://mosquitto.org/

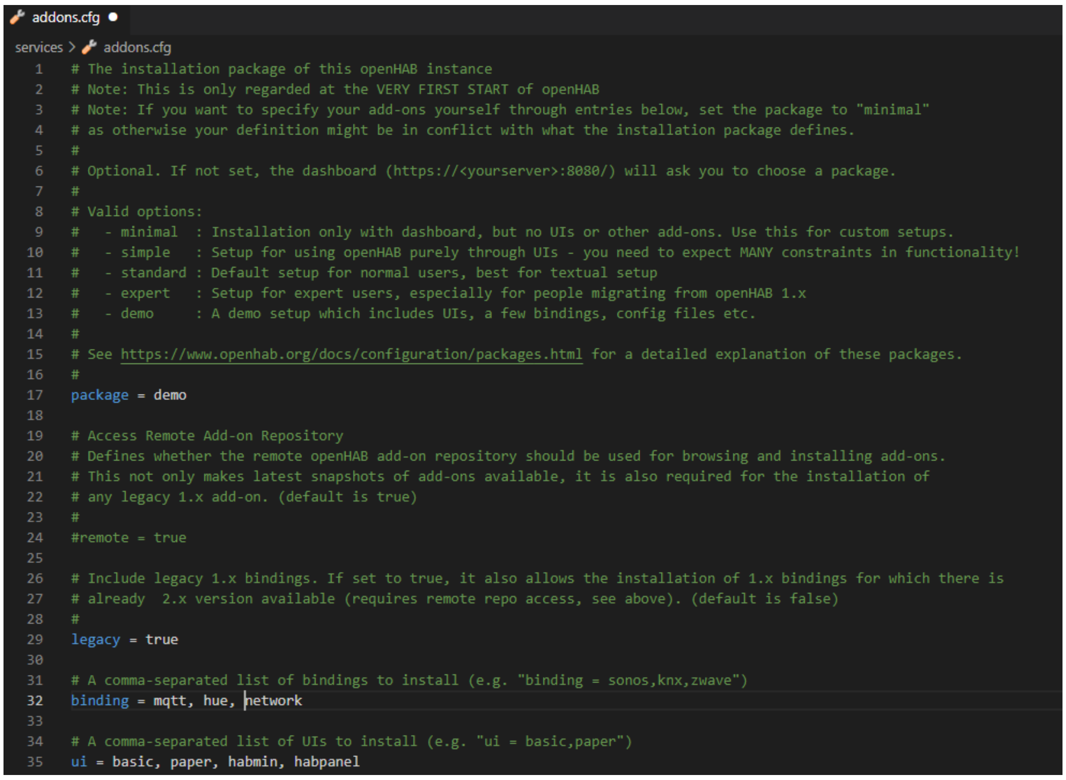
Add-ons, as described on openHAB add-ons page [17], are the base components responsible for integrating support for devices and services, integrating external systems, handling data storage and transformation, extending the automation engine, and enabling voice features. Each add-on for device or service contains tools for openHAB to monitor, control and automate them.

Things are the base components representing all the entities like devices, services, etc., that are managed by the system. According to documentation [18], they are connected to the system through the add-ons, which enable the system to manage them. The system can access the devices or services functionality through Channels that every corresponding Thing has.

Item is described in the documentation [19] as the base component that can represent all the properties of the automation system. They can be strings, number, switches, sliders, colour pickers or other Item types. It is mentioned, that item can be connected to a Thing to have control over its Channel. Items can be used in both automation and when defining user interfaces as they enable interaction with Things with corresponding devices and services that they are connected to.

First, the add-on for that device needs to be installed to add a device to the system. This can be done by modifying the *addons.cfg* file by adding the add-on package name to bindings list, as shown in Figure 9 or by installing through Paper UI, as shown in Figure 10. Suppose this process is done by modifying the *addons.cfg* file, then the package name can be found on the Add-ons page[9] by searching for the device or service. After a successful search, the result page can be opened, and the required package name is the last slash-separated word in the URL, as shown in Figure 11.

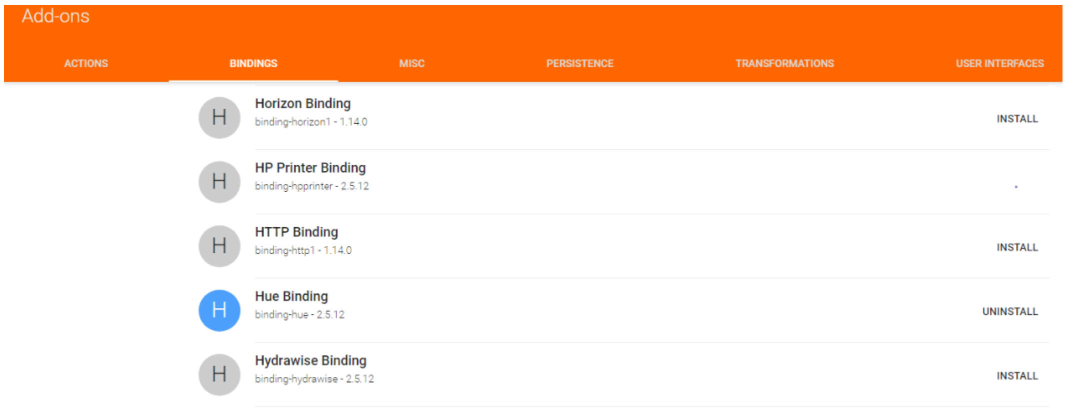
[9] https://www.openhab.org/addons/


addons.cfg ●

services > addons.cfg

```
# The installation package of this openHAB instance
# Note: This is only regarded at the VERY FIRST START of openHAB
# Note: If you want to specify your add-ons yourself through entries below, set the package to "minimal"
# as otherwise your definition might be in conflict with what the installation package defines.
#
# Optional. If not set, the dashboard (https://<yourserver>:8080/) will ask you to choose a package.
#
# Valid options:
#   - minimal  : Installation only with dashboard, but no UIs or other add-ons. Use this for custom setups.
#   - simple   : Setup for using openHAB purely through UIs - you need to expect MANY constraints in functionality!
#   - standard : Default setup for normal users, best for textual setup
#   - expert   : Setup for expert users, especially for people migrating from openHAB 1.x
#   - demo     : A demo setup which includes UIs, a few bindings, config files etc.
#
# See https://www.openhab.org/docs/configuration/packages.html for a detailed explanation of these packages.
#
package = demo

# Access Remote Add-on Repository
# Defines whether the remote openHAB add-on repository should be used for browsing and installing add-ons.
# This not only makes latest snapshots of add-ons available, it is also required for the installation of
# any legacy 1.x add-on. (default is true)
#
#remote = true

# Include legacy 1.x bindings. If set to true, it also allows the installation of 1.x bindings for which there is
# already  2.x version available (requires remote repo access, see above). (default is false)
#
legacy = true

# A comma-separated list of bindings to install (e.g. "binding = sonos,knx,zwave")
binding = mqtt, hue, network

# A comma-separated list of UIs to install (e.g. "ui = basic,paper")
ui = basic, paper, habmin, habpanel
```

Figure 9. *addons.cfg* file with instructions and listed installed add-ons.


Add-ons

ACTIONS BINDINGS MISC PERSISTENCE TRANSFORMATIONS USER INTERFACES

H	Horizon Binding binding-horizon1 - 1.14.0	INSTALL
H	HP Printer Binding binding-hpprinter - 2.5.12	
H	HTTP Binding binding-http1 - 1.14.0	INSTALL
H	Hue Binding binding-hue - 2.5.12	UNINSTALL
H	Hydrawise Binding binding-hydrawise - 2.5.12	INSTALL

Figure 10. Add-ons menu in Paper UI tool.

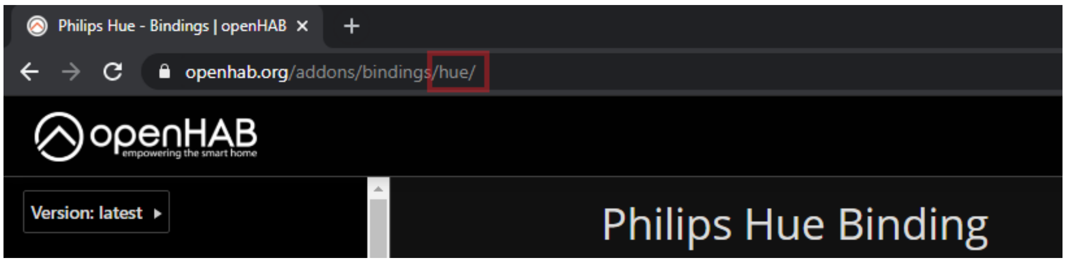

Philips Hue - Bindings | openHAB
openhab.org/addons/bindings/hue/
openHAB
empowering the smart home
Version: latest
Philips Hue Binding

Figure 11. Example of finding package name for *addons.cfg* file.

If an add-on was installed through Paper UI, then after a system restart, these add-ons are removed and need to be added again. This is not the case when add-ons are installed by modifying the add-ons configuration file, and as such, this method is used in this thesis for installing add-ons.

In this thesis, as the only smart devices, we had access to were Philips Hue smart lights, the add-on was called Philips Hue Binding with package name "hue". For services, we used an add-on for MQTT with the package name "mqtt". It was added to the bindings list in *add-ons.cfg* file.

After support for devices and services is enabled for openHAB by installing corresponding add-ons, the next step is to add devices and services. This can be done by specifying them in the Things file, *default.things*, manually or by automatically searching for them in Paper UI. If done manually by modifying the Things file for devices and services, it is advised to follow instructions given by the corresponding add-on on the add-ons page.

In this thesis, the devices were added through the usage of Paper UI as this process seemed quicker and easier. To add the devices on the Paper UI, we first went to the "Inbox" menu and clicked the "Scan" button, as seen in Figure 12. After few moments, devices started to appear. The Philips Hue lights that we had were connected to Hue Bridge. Both the Bridge and each light were added by clicking on the blue circle with white checkmark and then on "ADD AS THING" button, as shown in Figure 13 and 14.

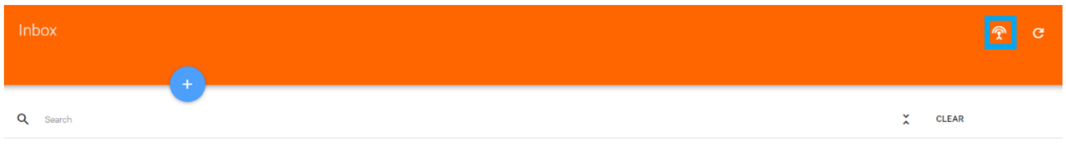
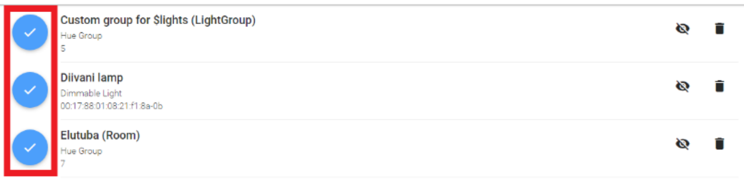


Figure 12. Scanning for devices on network through Inbox menu.

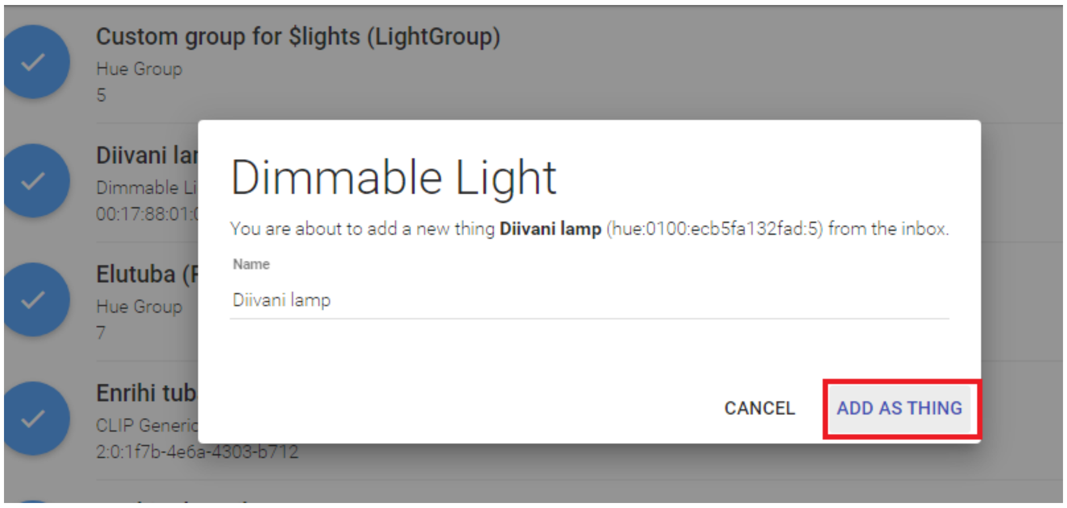


Custom group for $lights (LightGroup)
Hue Group
5

Diivani lamp
Dimmable Light
00:17:88:01:08:21:f1:8a-0b

Elutuba (Room)
Hue Group
7

Figure 13. Buttons for adding found devices on network.



Dimmable Light

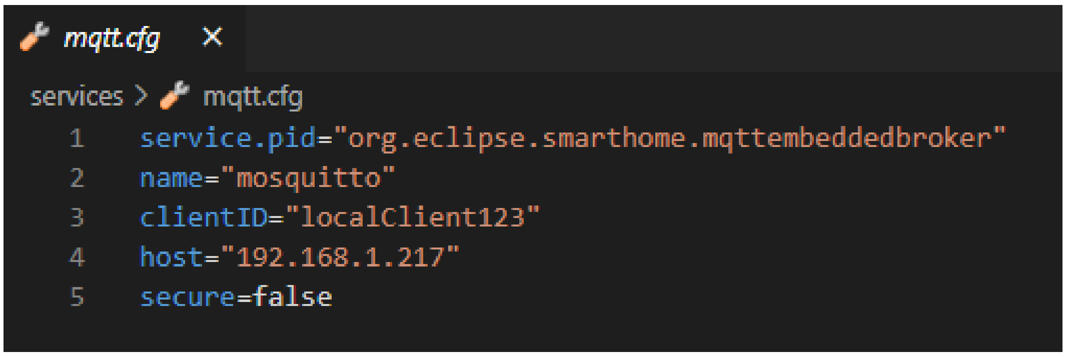
You are about to add a new thing **Diivani lamp** (hue:0100:ecb5fa132fad:5) from the inbox.

Name: Diivani lamp

Figure 14. Confirmation pop-up for adding a device.

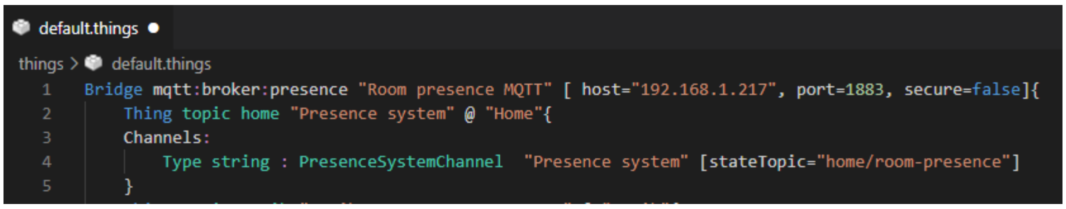
The MQTT service was added by modifying Things file *default.things* and adding additional configuration file *mqtt.cfg*. This configuration file was for holding core parameters for the

MQTT service. The Things file was for defining the MQTT service as an entity so that openHAB could access its functionalities. For the MQTT service, a broker and a client were determined, where Channel for the client was added for storing data from the subscribed topic. Figures 15 and 16 show the contents of described files.



```
service.pid="org.eclipse.smarthome.mqttembeddedbroker"
name="mosquitto"
clientID="localClient123"
host="192.168.1.217"
secure=false
```

Figure 15. Example of Mosquitto MQTT configuration file.



```
Bridge mqtt:broker:presence "Room presence MQTT" [ host="192.168.1.217", port=1883, secure=false]{
    Thing topic home "Presence system" @ "Home"{
    Channels:
        Type string : PresenceSystemChannel  "Presence system" [stateTopic="home/room-presence"]
    }
```

Figure 16. MQTT configured as a Thing in *default.things* file.

4.2.3.3 Automation

According to openHAB documentation, automations in openHAB have their base component known as Rules. Each rule works through a lightweight rule engine and invokes a script when triggered. Rules can be defined either through Paper UI or by writing rule files, where the file could hold multiple rules. Rule syntax is based on Xbase and follows the structure of:

- Rule name – a unique name for each rule
- Trigger condition – an event that triggers the rule execution
- Script block – a container for logic that should be executed on the trigger

For executing a rule, there are different categories of triggers:

- Item event-based – The Item-based triggers react to Item updates
- Thing event-based – Thing based triggers react to device or service status changes
- Group event-based – Group based triggers react to Item state changes that are in a specific group
- Time event passed – Time-based triggers react at specified times
- System event-based – The System based triggers react to system start events

Rules after triggering can do various things. In the documentation [10] it is mentioned, that they could change Item states, send command to Things, accomplish mathematical calculations and even trigger other scripts.

The automation that is implemented in this thesis is for controlling devices that are in the same room as the user. In essence, as the only smart devices that we have for this thesis are the Philips Hue smart lights, then if the user's location is near a room, where light is, then the light in that room will turn on. Similarly, if the user leaves the room, then the light in there turns off.

This automation is accomplished through a rule that triggers whenever the MQTT client that is subscribed to a specific topic receives new data. This data is in JSON format, containing information about:

- which room this data came from
- Bluetooth signal strength between each users device and the tracker
- if any guest is in the room

This rule uses the data to determine if any user moved to another room by comparing the signal strengths between the current room and the last room. If the current room is different from the previous room and the signal strength is better, then the user is written into the new room. This rule also writes user out of any room if their Bluetooth signal can not be determined.

After all user locations are determined, the rule will check if any room user count reached zero or became greater than zero. In case it reached zero, then Philips Hue lights in that room are turned off. But if the user count changes from zero to a greater value, then lights in that room are turned on.

Additionally, a switch disables or enables this rule, which users can manually control through a user interface.

The described rule will be shown in Appendix I.

4.3 User Tracking System Configuration

This section will focus on the user tracking system. The following subsections will cover the overview of the system, the setup process and finally, explain the script used for user tracking.

4.3.1 System

The user tracking system will consist of multiple individual trackers. Each tracker will be placed in different rooms and will independently track user whereabouts through Bluetooth scanning. This information is then sent to the home automation platform openHAB through MQTT. For trackers to accomplish such actions, they need to have support for WIFI and Bluetooth. Other than that, they also need to be able to run the script that scans for users, compiles the data and sends it over the MQTT.

As mentioned in the Background section of this thesis, Arduino microcontrollers and Raspberry Pi are single-board computers available for such tasks. For this thesis, Raspberry Pi Zero W will be used for each tracker. The main reasons for picking Raspberry over the Arduino was that they allow for better experimenting with different software and scripting languages and are more accessible.

According to the official Raspberry product specification [20], the used Raspberry Pi Zero W's have the following specifications:

- CPU – Single-core, 1 GHz
- RAM – 4GB LPDDR4-3200 SDRAM
- WIFI – 2.4 GHz 802.11 b/g/n wireless
- Bluetooth – Bluetooth 4.1 and BLE

Each Raspberry Pi Zero W will be running Linux based Raspberry Pi OS Lite that is a port of Debian. This operating system does not have graphical output and works as a headless system.

4.3.2 Initial Setup

With the hardware and operating system introduced in the previous section, each tracker can be set up.

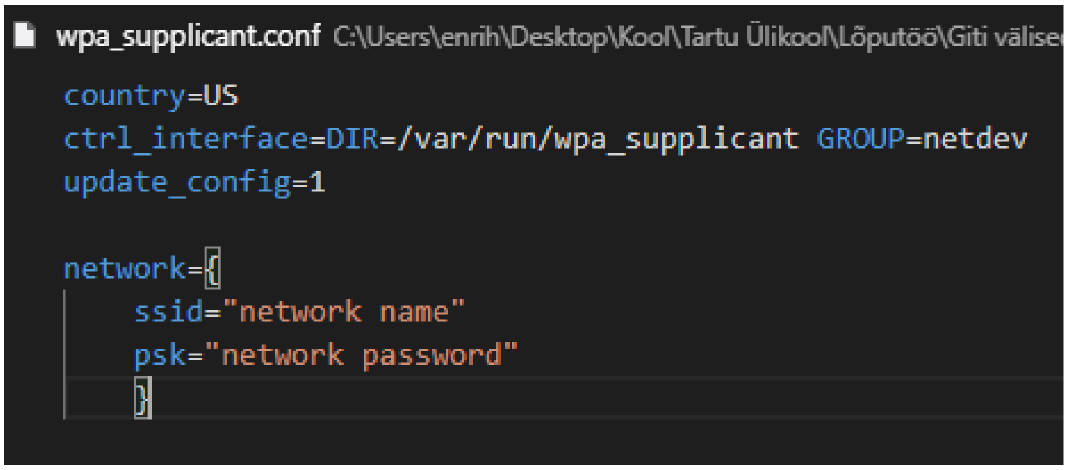
Trackers, that were set up in this thesis required the following components:

- Raspberry Pi Zero W
- Raspberry Pi OS Lite image
- Flash tool
- Micro-SD card with 4 GB or more storage
- Micro-SD card reader
- WIFI access

To get the Raspberry Pi OS Lite running on Raspberry Pi Zero W, the image file is first required. For this, a tool like Raspberry Pi Imager[10] can be used to acquire the image file and flash the image onto a Micro-SD card. As with the home automation platform setup, to flash the image onto a Micro-SD card, a Micro-SD card reader is necessary.

After the image is acquired and flashed onto a Micro-SD card, the image should be configured to connect to WIFI after booting up and allowing SSH connection. For that image root directory needs to be accessed. For enabling SSH, an empty file named *ssh* should be created. As for enabling access to WIFI, a file named *wpa_supplicant.conf* should be created with the following contents that can be found in Figure 17.

[10] https://www.raspberrypi.org/software/



```
wpa_supplicant.conf C:\Users\enrih\Desktop\Kool\Tartu Ülikool\Lõputöö\Giti välise
country=US
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="network name"
    psk="network password"
}
```

Figure 17. Example of *wpa_supplicant.conf* file.

After configuration for SSH and WIFI is done, the Micro-SD card can be mounted into the Raspberry Pi Zero W and booted up. It can take a couple of minutes for trackers to boot up entirely, but once they appear on the network routers devices list, an SSH session can be created to connect to the trackers.

When the SSH session is created, the console will prompt the user to log in. The default username is "pi", and the password is "raspberry". After that, access to the Raspberry Pi OS system is obtained. As Raspberry Pi OS is Linux based system, then the command line console can be operated with Linux commands.

For the user tracking script to work, additional packages need to be installed. First, the modules for MQTT need to be installed. For this, the following commands need to be run:

- sudo apt-get install mosquito
- sudo apt-get install mosquitto-clients

After that, a module with Bluetooth tools need to be installed through the following command:

- sudo apt-get install bluetooth bluez libbluetooth-dev

Now that the modules for sending data through MQTT and working with Bluetooth are installed next, the tools necessary for running the Python script are needed. This Python script will use the *paho-mqtt*[11] module to allow the use of MQTT resources and *pybluez*[12] module to enable the use of Bluetooth resources in the script. Python and mentioned Python modules can be installed through the following commands:

- sudo apt-get install python-pip
- sudo python -m pip install paho-mqtt
- sudo python -m pip install pybluez

It should be noted that the Python version that is used by this script is Python 2. Although Python 2 is deprecated, the *pybluez* module did not work well with Python 3.

[11] https://pypi.org/project/paho-mqtt/
[12] https://pypi.org/project/PyBluez/

After all the required packages are installed, the script for user tracking can be implemented.

4.3.3 User Tracking Software

The user tracking script has multiple tasks defined. These tasks are for handling MQTT service, scanning Bluetooth devices, measuring their signal strength from the tracker, and compiling and sending acquired data through MQTT.

The MQTT has tasks to handle the connection and disconnection. In case of disconnecting from the broker function, *signal_handler* will be triggered, which will gracefully close the MQTT client and end the script. As for handling the connection, the function *on_connect* will trigger when the connection to the broker is attempted. After activating, the function will print out the connection result code. This function can be used in the future to trigger specific actions, but as of now lacks any functionality.

For scanning for the Bluetooth devices and measuring their signal strength, there are functions *guests_nearby* and *bluetooth_rssi*. The first function will try to discover new devices, and if a device with strong enough signal strength is found, the function returns true, otherwise, false. The second function will look for the predefined device and measure their signal strength, which will be returned at the end. This function was forked from GitHub user dagar[13].

The main function works by first setting up MQTT and connecting to the broker on the home automation platform. After that, a loop will look for guests for a certain amount of time, then acquiring signal strength from each predefined user device. When this is done, the data will be packaged into the JSON format and sent by MQTT publish action. After that loop repeats the process.

All the explained functions are shown in Appendix II.

[13] https://github.com/dagar/bluetooth-proximity

4.4 Example of implementation

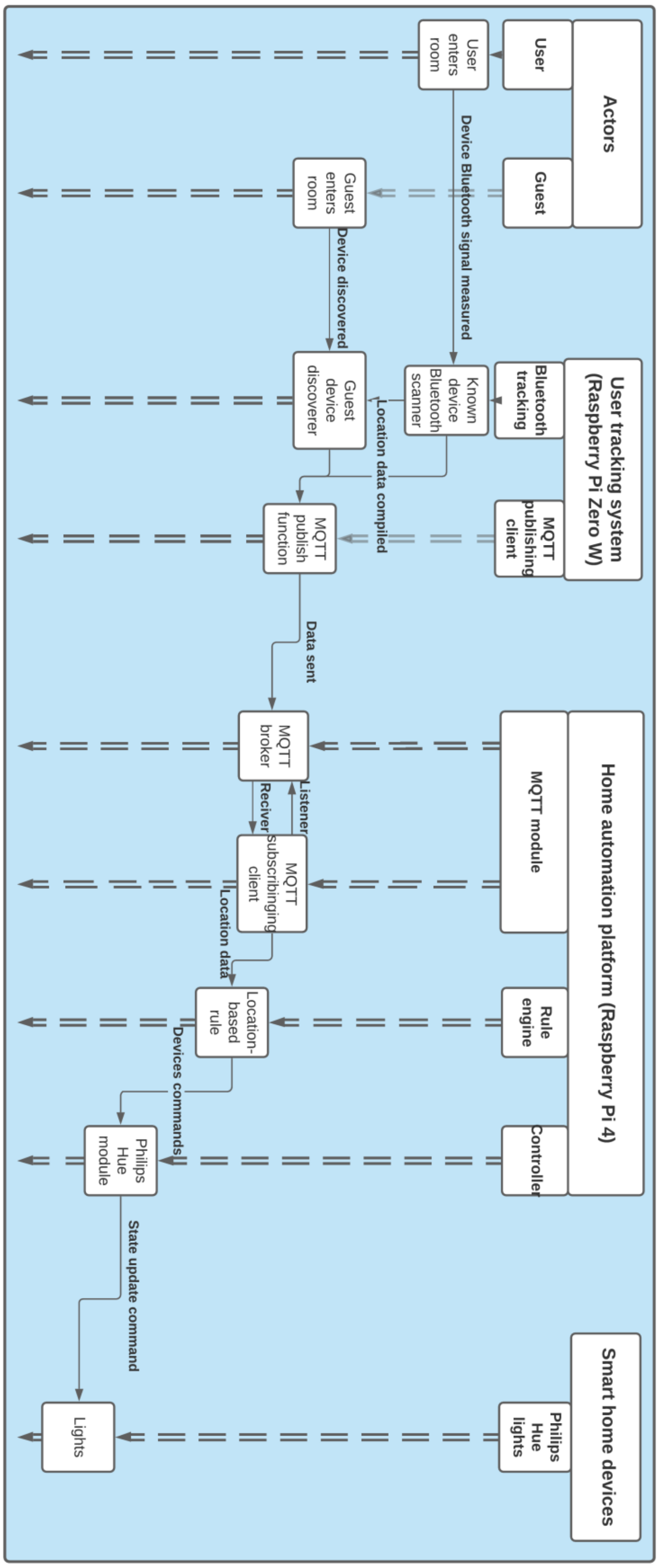


Figure 18. Sequence diagram of the smart home system’s workflow.

The smart home system's workflow from detecting users location to controlling smart devices around the home through automation is explained in Figure 18. The workflow goes through four major groups, each containing one or more minor components.

The first group is the Actors. This group contains the different actors, that interact with the system. They are the users and the guests and can interact with the system by moving around the house, from one room to another.

The second group is the user tracking system. This diagram represents the individual trackers in different rooms. This group has two components, the Bluetooth tracking module and the MQTT publishing client. Bluetooth tracking module scans for devices nearby and pickups the known and unknown nearby devices from users and guests. Each discovered device signal strength is measured and compiled into location data. This data also contains information about guests and the room the tracker is in. After the data is compiled, then the MQTT publishing client sends the data to the home automation systems MQTT broker.

The third group is the home automation platform. This group contains three components, the MQTT module, the rule engine and the controller. The MQTT module is responsible for handling MQTT broker and subscribing client. The subscribing MQTT client is listening to MQTT broker and once new data is available it receives it. From there the data is forwarded to rule engine.

The rule engine is responsible for handling automations and can contain multiple rules. The location-based automation rule implemented in this thesis is triggered whenever the subscribing MQTT client receives new data. The rule takes the received data and process it. After that it will use this data to figure out which devices should be turned off or on and sends this information to the controller. The controller uses this data to send commands to each concerning device.

The last group is for smart home devices. This contains all the different devices connected to the home automation platform. These devices can be controlled by the commands issued by the controller from the home automation platform.

5. Implementation Challenges

When implementing the home automation system and creating the user tracking system, there were few problems that needed to be resolved. The more significant issues on openHAB were file system corrupting and MQTT broker frequent crashes. As for the tracking system, the main problem was with IP address changing.

The most severe problem was the file system corruption on the openHAB system. This occurred after the power was cut from the openHAB system without prior safe shut down. As openHABian is Linux based system, then ungraceful shutdowns like this can easily break the file system. To overcome this, there are a couple of ways. One of them is to have a UPS, which could power the system for a short time after power is cut. During this time, the UPS could send a shutdown signal to the openHAB system and prevent file systems from corrupting.

The other way is to occasionally back up the system on an external storage device. In the event of corruption occurring, the system could be easily restored from the external backup. This solution was used to prevent setting up and reconfiguring a new system, should it happen again.

The problem with MQTT crashing on openHAB could not be traced to a cause. This problem seemed to be mainly because multiple clients were publishing data to broker and overwhelming it in the process. To solve this, the user tracking scripts main loop was slowed down. After that, MQTT on openHAB seemed to work fine.

The only problem faced with trackers was that their IP address could change when they were rebooted. This problem made it difficult to access the trackers through an SSH session, as the new IP address was necessary. There were multiple ways of solving this problem, but the easiest way was to modify these devices on the network router and assign a permanent IP address to them.

6. Conclusion

This thesis aimed to create a smart home system that could monitor and control smart home appliances through IoT. As a result of this thesis, an IoT system was built to manage these devices and achieve automation using data sent through an IoT-friendly communication protocol. This system consisted of two sub-systems, the home automation platform and the user tracking system.

The sub-system responsible for managing the home through monitoring, controlling and automating smart home devices and services was the home automation platform. This platform was implemented using an open-source smart home platform called openHAB. It allowed for connecting and managing multiple different devices and services from diverse ecosystems and could be run on single-board computers like Raspberry Pi. As such, the Raspberry Pi 4 was used to run the home automation platform.

To demonstrate the IoT capabilities in a smart home environment, the IoT friendly messaging protocol MQTT was used to enable the two sub-systems to communicate. The home automation platform housed a Mosquitto MQTT broker and subscriber client, where the subscriber client logged the user location data that was sent over the IoT network. This data was used by the automation rule to switch Philips Hue lights on and off, based on the users' proximity to them.

To acquire the users' location in the house, the user tracking system, which was developed and created in this thesis, consisted of individual tracking devices placed in different rooms around the house. These tracker devices were built by using Raspberry Pi Zero W with a user tracking Python script. This model of Raspberry Pi single-board computers could use WIFI and Bluetooth and run a lightweight operating system. That operating system hosted the Mosquitto MQTT service and ran the Python script responsible for collecting Bluetooth data and publishing it through the MQTT publishing client.

Both of the sub-systems create the whole smart home IoT system that monitors and controls smart home appliances. The configuration, automation and script files for both systems are located on GitHub repository page [21].

7. References

[1] Research and Markets, “IoT Middleware Market - Growth, Trends, and Forecasts (2020 - 2025),” September 2020. [Online]. Available: https://www.researchandmarkets.com/reports/5174853/iot-middleware-market-growth-trends-and. [Accessed 15 March 2021].

[2] Adroit Market Research, "Smart Home Automation Market," November 2020. [Online]. Available: https://www.adroitmarketresearch.com/industry-reports/smart-home-automation-market. [Accessed 10 December 2020].

[3] Home Assistant, “Integrations,” [Online]. Available: https://www.home-assistant.io/integrations/. [Accessed 23 March 2021].

[4] A. Williams, “Home Assistant lets you automate your smart home without giving up privacy,” 10 May 2018. [Online]. Available: https://www.the-ambient.com/features/home-assistant-automation-privacy-582. [Accessed 31 March 2021].

[5] P. Schoutsen, “Perfect Home Automation,” 19 January 2016. [Online]. Available: https://www.home-assistant.io/blog/2016/01/19/perfect-home-automation/. [Accessed 31 March 2021].

[6] Home Assistant, “Documentation,” [Online]. Available: https://www.home-assistant.io/docs/. [Accessed 31 March 2021].

[7] openHAB Foundation, “Introduction,” [Online]. Available: https://www.openhab.org/docs/. [Accessed 31 March 2021].

[8] K. Kreuzer, “openHAB 2.0 and Eclipse SmartHome,” 16 June 2014. [Online]. Available: http://www.kaikreuzer.de/2014/06/16/openhab-20-and-eclipse-smarthome/#ug. [Accessed 31 March 2021].

[9] openHAB Foundation, “Who We Are,” [Online]. Available: https://www.openhab.org/about/who-we-are.html. [Accessed 31 March 2021].

[10] openHAB Foundation, “Textual Rules,” [Online]. Available: https://www.openhab.org/docs/configuration/rules-dsl.html. [Accessed 31 March 2021].

[11] Apple Inc., “HomeKit,” [Online]. Available: https://developer.apple.com/homekit/. [Accessed 17 April 2021].

[12] Apple Inc., “Home,” [Online]. Available: https://www.apple.com/ios/home/. [Accessed 17 April 2021].

[13] J. Porter, “HomeKit might be fading, but Apple’s not giving up yet,” 28 October 2019. [Online]. Available: https://www.theverge.com/2019/10/28/20936292/apple-homekit-hiring-engineers. [Accessed 17 April 2021].

[14] Arduino, “Introduction,” [Online]. Available: https://www.arduino.cc/en/guide/introduction. [Accessed 20 April 2021].

[15] Signify Holding, “Explore Us,” [Online]. Available: https://www.philips-hue.com/en-us/explore-hue/how-it-works. [Accessed 12 April 2021].

[16] openHAB Foundation, “Configuration Overview,” [Online]. Available: https://www.openhab.org/docs/configuration/. [Accessed 27 April 2021].

[17] openHAB Foundation, “Add-Ons,” [Online]. Available: https://www.openhab.org/addons/. [Accessed 27 April 2021].

[18] openHAB Foundation, “Things,” [Online]. Available: https://www.openhab.org/docs/configuration/things.html. [Accessed 27 April 2021].

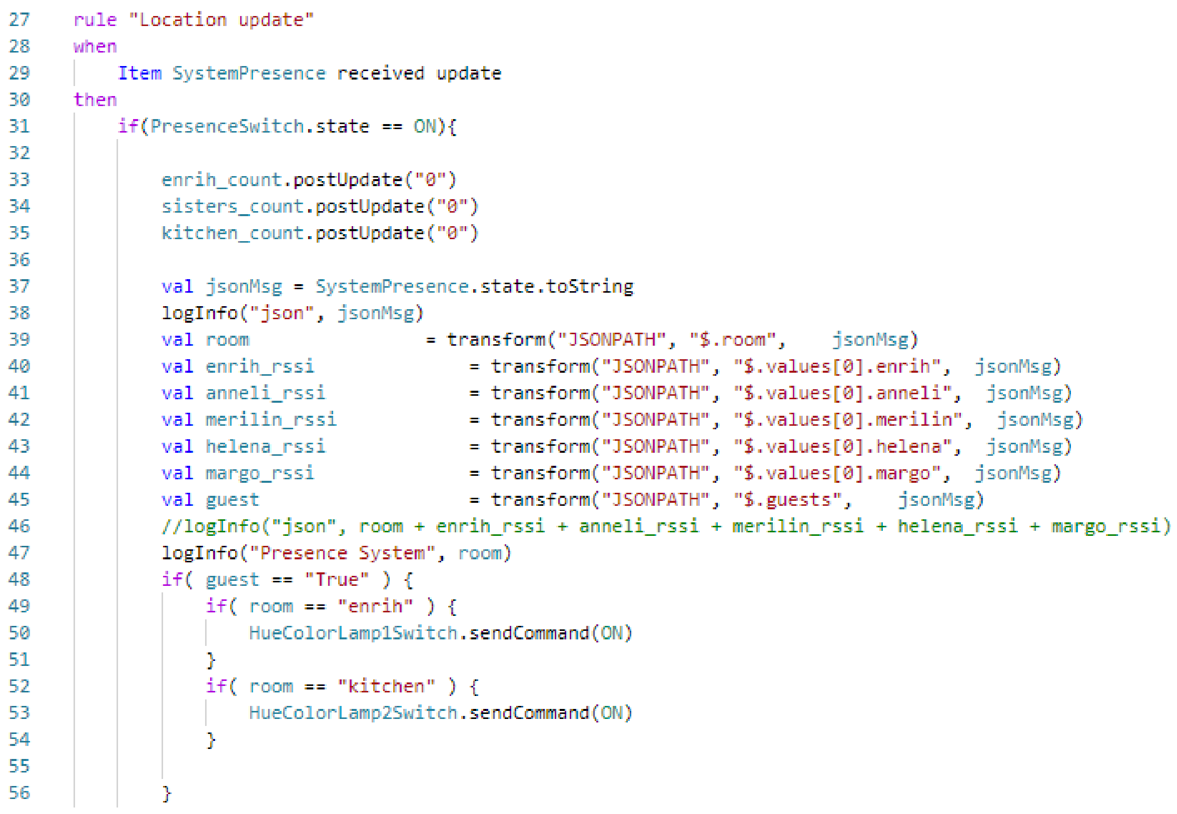
[19] openHAB Foundation, “Items,” [Online]. Available: https://www.openhab.org/docs/configuration/items.html. [Accessed 27 April 2021].

[20] Raspberry Pi Foundation, “Raspberry Pi Zero W,” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-zero-w/. [Accessed 27 April 2021].

[21] E. Sinilaid, “Monitoring and controlling smart home appliances using IoT devices,” 5 May 2021. [Online]. Available: https://github.com/EnrihSinilaid/Monitoring-and-controlling-smart-home-appliances-using-IoT-devices. [Accessed 5 May 2021].

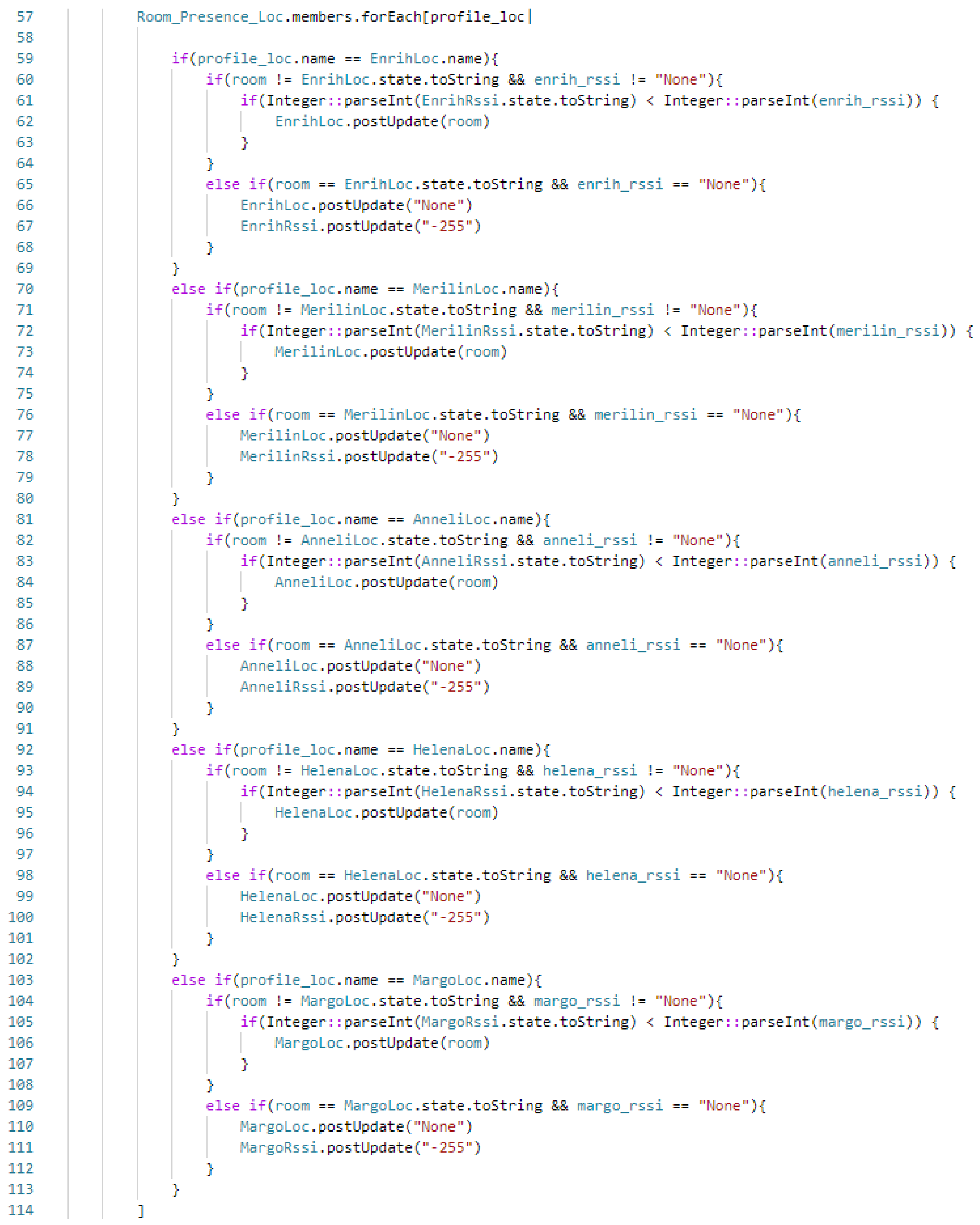
Appendix

I. Rule for user location based automation



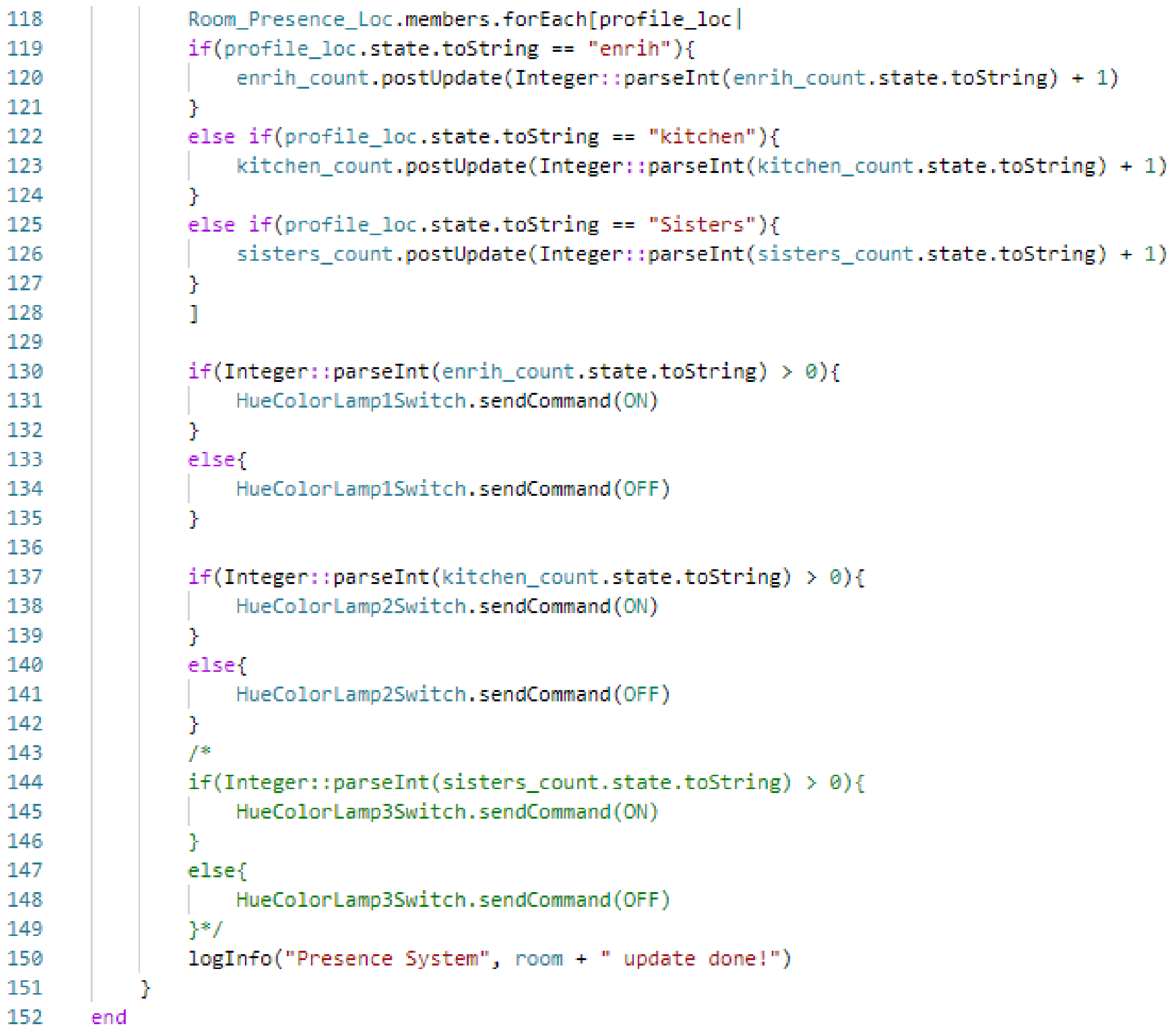
```
rule "Location update"
when
    Item SystemPresence received update
then
    if(PresenceSwitch.state == ON){

        enrih_count.postUpdate("0")
        sisters_count.postUpdate("0")
        kitchen_count.postUpdate("0")

        val jsonMsg = SystemPresence.state.toString
        logInfo("json", jsonMsg)
        val room                = transform("JSONPATH", "$.room",    jsonMsg)
        val enrih_rssi              = transform("JSONPATH", "$.values[0].enrih",  jsonMsg)
        val anneli_rssi             = transform("JSONPATH", "$.values[0].anneli",  jsonMsg)
        val merilin_rssi            = transform("JSONPATH", "$.values[0].merilin",  jsonMsg)
        val helena_rssi             = transform("JSONPATH", "$.values[0].helena",  jsonMsg)
        val margo_rssi              = transform("JSONPATH", "$.values[0].margo",  jsonMsg)
        val guest                   = transform("JSONPATH", "$.guests",    jsonMsg)
        //logInfo("json", room + enrih_rssi + anneli_rssi + merilin_rssi + helena_rssi + margo_rssi)
        logInfo("Presence System", room)
        if( guest == "True" ) {
            if( room == "enrih" ) {
                HueColorLamp1Switch.sendCommand(ON)
            }
            if( room == "kitchen" ) {
                HueColorLamp2Switch.sendCommand(ON)
            }

        }
```

Figure 19. Rules first part, processing location data.



```
        Room_Presence_Loc.members.forEach[profile_loc|

            if(profile_loc.name == EnrihLoc.name){
                if(room != EnrihLoc.state.toString && enrih_rssi != "None"){
                    if(Integer::parseInt(EnrihRssi.state.toString) < Integer::parseInt(enrih_rssi)) {
                        EnrihLoc.postUpdate(room)
                    }
                }
                else if(room == EnrihLoc.state.toString && enrih_rssi == "None"){
                    EnrihLoc.postUpdate("None")
                    EnrihRssi.postUpdate("-255")
                }
            }
            else if(profile_loc.name == MerilinLoc.name){
                if(room != MerilinLoc.state.toString && merilin_rssi != "None"){
                    if(Integer::parseInt(MerilinRssi.state.toString) < Integer::parseInt(merilin_rssi)) {
                        MerilinLoc.postUpdate(room)
                    }
                }
                else if(room == MerilinLoc.state.toString && merilin_rssi == "None"){
                    MerilinLoc.postUpdate("None")
                    MerilinRssi.postUpdate("-255")
                }
            }
            else if(profile_loc.name == AnneliLoc.name){
                if(room != AnneliLoc.state.toString && anneli_rssi != "None"){
                    if(Integer::parseInt(AnneliRssi.state.toString) < Integer::parseInt(anneli_rssi)) {
                        AnneliLoc.postUpdate(room)
                    }
                }
                else if(room == AnneliLoc.state.toString && anneli_rssi == "None"){
                    AnneliLoc.postUpdate("None")
                    AnneliRssi.postUpdate("-255")
                }
            }
            else if(profile_loc.name == HelenaLoc.name){
                if(room != HelenaLoc.state.toString && helena_rssi != "None"){
                    if(Integer::parseInt(HelenaRssi.state.toString) < Integer::parseInt(helena_rssi)) {
                        HelenaLoc.postUpdate(room)
                    }
                }
                else if(room == HelenaLoc.state.toString && helena_rssi == "None"){
                    HelenaLoc.postUpdate("None")
                    HelenaRssi.postUpdate("-255")
                }
            }
            else if(profile_loc.name == MargoLoc.name){
                if(room != MargoLoc.state.toString && margo_rssi != "None"){
                    if(Integer::parseInt(MargoRssi.state.toString) < Integer::parseInt(margo_rssi)) {
                        MargoLoc.postUpdate(room)
                    }
                }
                else if(room == MargoLoc.state.toString && margo_rssi == "None"){
                    MargoLoc.postUpdate("None")
                    MargoRssi.postUpdate("-255")
                }
            }
        ]
```

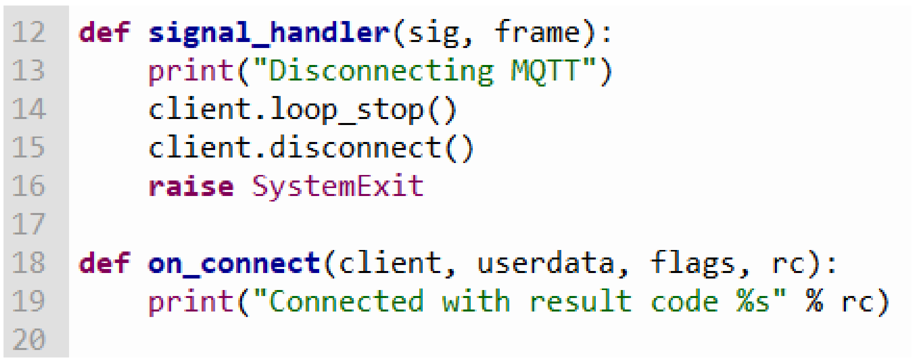
Figure 20. Rules second part, assigning rooms to users.



```
        Room_Presence_Loc.members.forEach[profile_loc|
        if(profile_loc.state.toString == "enrih"){
            enrih_count.postUpdate(Integer::parseInt(enrih_count.state.toString) + 1)
        }
        else if(profile_loc.state.toString == "kitchen"){
            kitchen_count.postUpdate(Integer::parseInt(kitchen_count.state.toString) + 1)
        }
        else if(profile_loc.state.toString == "Sisters"){
            sisters_count.postUpdate(Integer::parseInt(sisters_count.state.toString) + 1)
        }
        ]

        if(Integer::parseInt(enrih_count.state.toString) > 0){
            HueColorLamp1Switch.sendCommand(ON)
        }
        else{
            HueColorLamp1Switch.sendCommand(OFF)
        }

        if(Integer::parseInt(kitchen_count.state.toString) > 0){
            HueColorLamp2Switch.sendCommand(ON)
        }
        else{
            HueColorLamp2Switch.sendCommand(OFF)
        }
        /*
        if(Integer::parseInt(sisters_count.state.toString) > 0){
            HueColorLamp3Switch.sendCommand(ON)
        }
        else{
            HueColorLamp3Switch.sendCommand(OFF)
        }*/
        logInfo("Presence System", room + " update done!")
    }
end
```

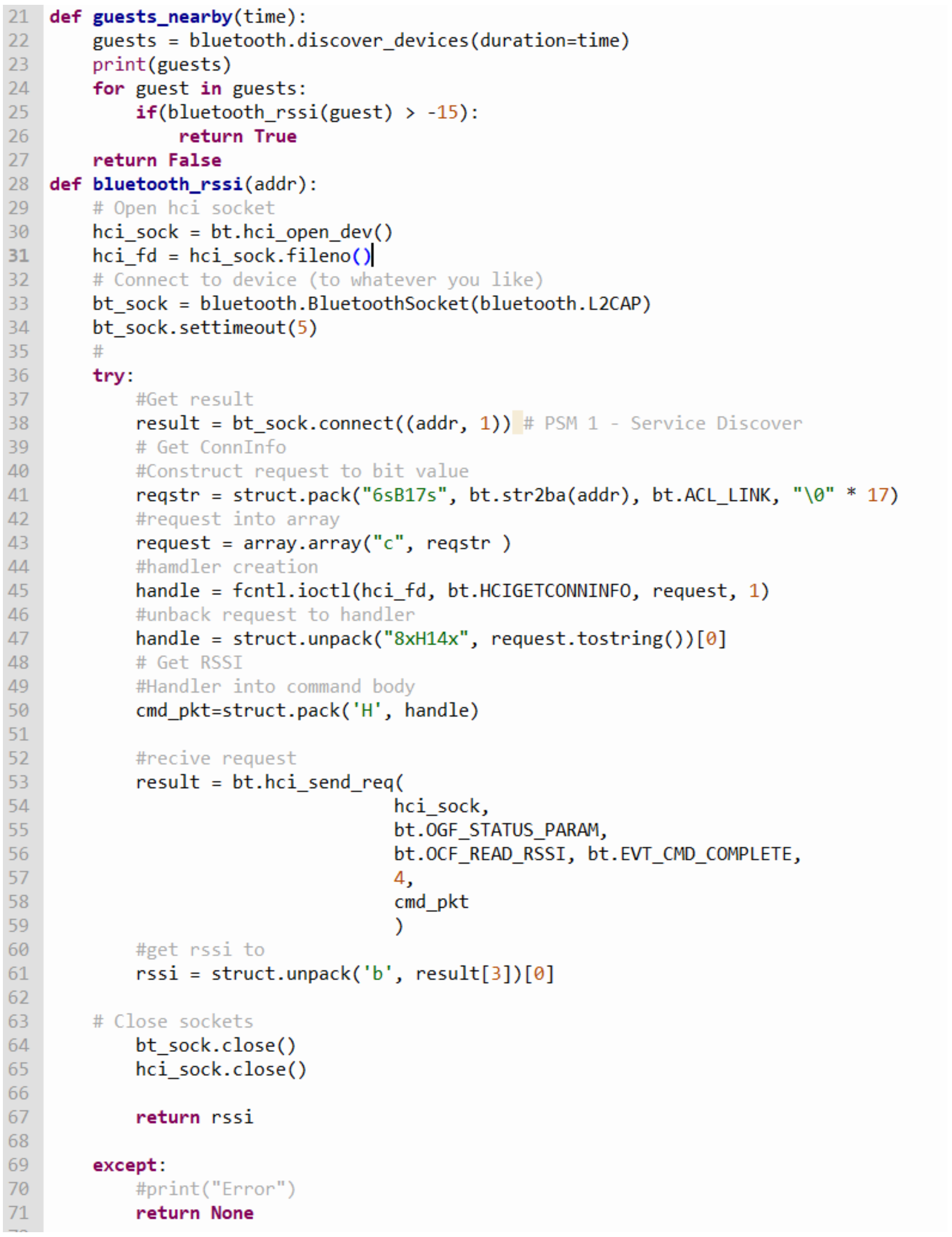
Figure 21. Rules third part, controlling rooms based on user locations.

II Custom user tracking script



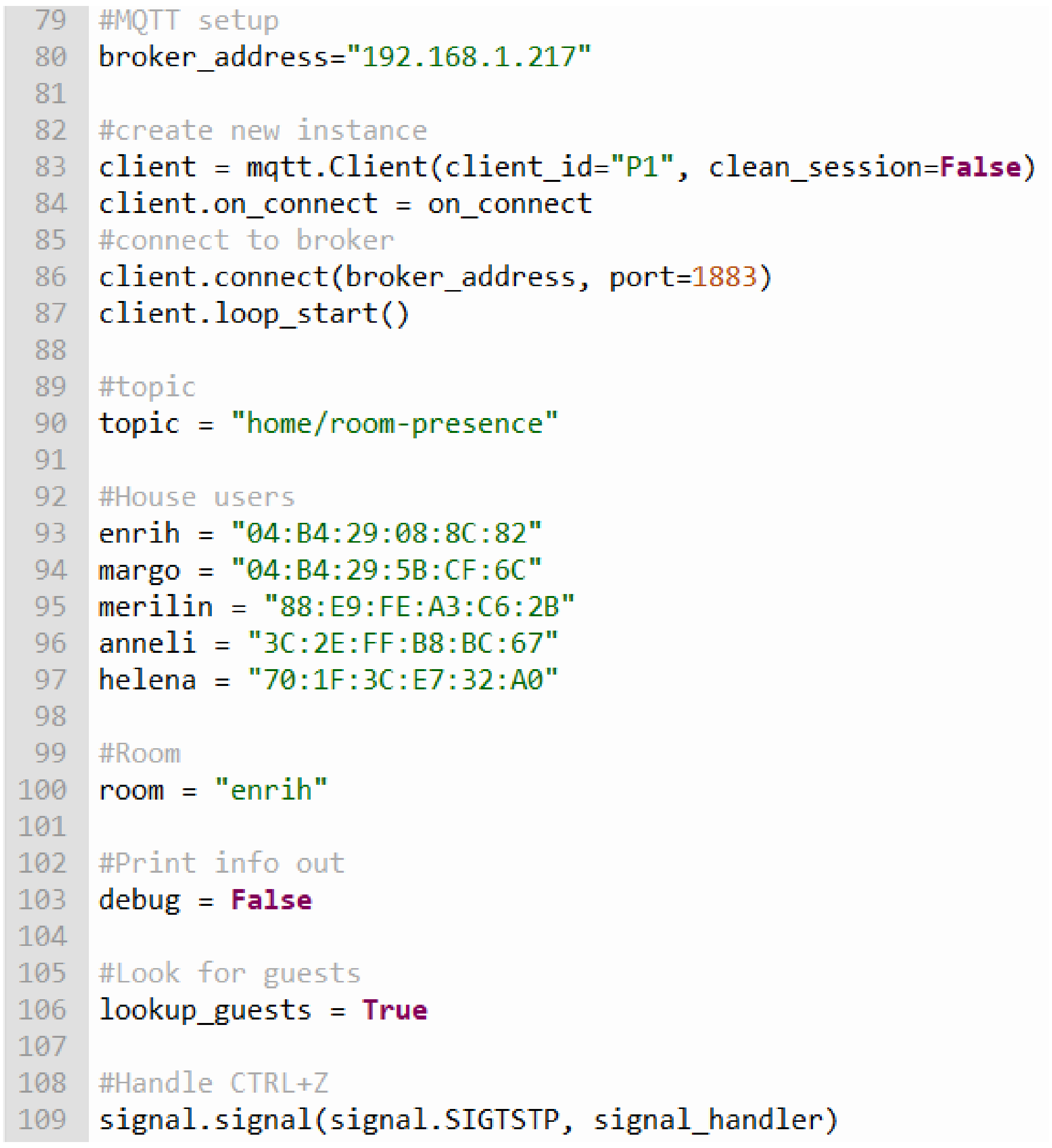
```python
def signal_handler(sig, frame):
    print("Disconnecting MQTT")
    client.loop_stop()
    client.disconnect()
    raise SystemExit

def on_connect(client, userdata, flags, rc):
    print("Connected with result code %s" % rc)

```

Figure 22. Functions *signal_handler* and *on_connection.*



```python
def guests_nearby(time):
    guests = bluetooth.discover_devices(duration=time)
    print(guests)
    for guest in guests:
        if(bluetooth_rssi(guest) > -15):
            return True
    return False
def bluetooth_rssi(addr):
    # Open hci socket
    hci_sock = bt.hci_open_dev()
    hci_fd = hci_sock.fileno()
    # Connect to device (to whatever you like)
    bt_sock = bluetooth.BluetoothSocket(bluetooth.L2CAP)
    bt_sock.settimeout(5)
    #
    try:
        #Get result
        result = bt_sock.connect((addr, 1)) # PSM 1 - Service Discover
        # Get ConnInfo
        #Construct request to bit value
        reqstr = struct.pack("6sB17s", bt.str2ba(addr), bt.ACL_LINK, "\0" * 17)
        #request into array
        request = array.array("c", reqstr )
        #hamdler creation
        handle = fcntl.ioctl(hci_fd, bt.HCIGETCONNINFO, request, 1)
        #unback request to handler
        handle = struct.unpack("8xH14x", request.tostring())[0]
        # Get RSSI
        #Handler into command body
        cmd_pkt=struct.pack('H', handle)

        #recive request
        result = bt.hci_send_req(
                                hci_sock,
                                bt.OGF_STATUS_PARAM,
                                bt.OCF_READ_RSSI, bt.EVT_CMD_COMPLETE,
                                4,
                                cmd_pkt
                                )
        #get rssi to
        rssi = struct.unpack('b', result[3])[0]

    # Close sockets
        bt_sock.close()
        hci_sock.close()

        return rssi

    except:
        #print("Error")
        return None
```

Figure 23. Functions *guest_nearby* and *bluetooth_rssi*.



```python
#MQTT setup
broker_address="192.168.1.217"

#create new instance
client = mqtt.Client(client_id="P1", clean_session=False)
client.on_connect = on_connect
#connect to broker
client.connect(broker_address, port=1883)
client.loop_start()

#topic
topic = "home/room-presence"

#House users
enrih = "04:B4:29:08:8C:82"
margo = "04:B4:29:5B:CF:6C"
merilin = "88:E9:FE:A3:C6:2B"
anneli = "3C:2E:FF:B8:BC:67"
helena = "70:1F:3C:E7:32:A0"

#Room
room = "enrih"

#Print info out
debug = False

#Look for guests
lookup_guests = True

#Handle CTRL+Z
signal.signal(signal.SIGTSTP, signal_handler)
```

Figure 24. First part of the main function.

```python
try:
    while True:

        #Guest present? Default = False
        guests = False

        #Look for guests if enabled
        if lookup_guests:
            guests = guests_nearby(5)

        # Get all home devices rssi values
        enrih_rssi = str(bluetooth_rssi(enrih))
        margo_rssi = str(bluetooth_rssi(margo))
        merilin_rssi = str(bluetooth_rssi(merilin))
        anneli_rssi = str(bluetooth_rssi(anneli))
        helena_rssi = str(bluetooth_rssi(helena))

        room_info = {
            "room": room,
            "guests": guests,
            "values": [
                {
                "enrih": enrih_rssi,
                "merilin": merilin_rssi,
                "helena": helena_rssi,
                "anneli": anneli_rssi,
                "margo": margo_rssi
                }
            ]
        }

        string_packet = json.dumps(room_info, separators=(',', ':'))

        client.publish(topic, string_packet)
        time.sleep(10)

        if(debug):
            print(string_packet)
            print("-----------")
            print(room_info)

except KeyboardInterrupt:
    pass
finally:
    print("End")
    client.loop_stop()
    client.disconnect()
```

Figure 25. Second part of the main function.

III License

I, **Enrih Sinilaid**

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work from 06.05.2021 until the expiry of the term of copyright,

"**Monitoring and controlling smart home appliances using IoT devices**",

supervised by **Chinmaya Kumar Dehury**

2. I am aware of the fact that the author retains the rights specified in p. 1.

3. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Enrih Sinilaid

Tallinn, 06.05.2021