UCLouvain

epl

École polytechnique de Louvain

Role-based access control for openHAB users

Author: **Nicolas Gennart**
Supervisors: **Etienne Riviere**, **Igor Zavalyshyn**
Readers: **Ramin Sadre**, **Annanda Rath**, **Yinan Cao**
Academic year 2021-2022
Master [120] in Cybersecurity

Acknowledgement

To my family and friends who supported and helped me make this thesis possible. Thank you.

To Maxime and Marc-André whose insights and proof reading were of great help. Thank you.

To Prof. Riviere and Dr. Zavalyshyn for their meaningful advice and guidance during the year. Thank you.

Contents

Acronyms

openHAB	open Home Automation Bus
IoT	Internet of Things
UI	User Interface
RBAC	Role-Based Access Control
DAC	Discretionary Access Control
MAC	Mandatory access control
ABAC	Attribute-based access control
VPN	Virtual Private Network

Chapter 1

Introduction

Open home automation bus (openHAB) is a technology-agnostic home automation platform to manage Internet of Things (IoT) devices in smart homes. OpenHAB is commonly used and its necessity increases fastly. This environment is developing rapidly and needs improvements in terms of access control and potential security breaches. This thesis intents to further study these two areas of improvement.

The first chapter is dedicated to the context of openHab and the shortage of access control. IoT devices, IoT ecosystems, and smart homes are first introduced. The OpenHAB platform managing these components will be explained. Why it is a good idea to choose OpenHAB over proprietary IoT platforms. The evaluation of open-source IoT platforms for smart homes by B. Setz, S. Graef, D. Ivanova A. Tiessen, and M. Aiello will be explained [76]. Despite the fact that openHAB is a robust home automation software, it has limited access control which will be illustrated through a scenario that is a smart family home. RBAC is the model chosen to address the lack of access control in the scenario. The goals will be stated to enable RBAC for openHAB users in a running instance of openHAB. Finally, the contribution made to openHAB will be detailed.

The objective of this second chapter is to have a better understanding of openHAB. The main concepts of openHAB will be defined. The general architecture of the home automation software will be illustrated. Then, the configuration of the smart family home which is our scenario will be explained step by step, with the semantic model, the syntax of the Items, the installed Bindings, and the configured Things, Channels, Items, and Links. Finally, the way the scenario can be reached and configured will be explained.

The main chapter is chapter three which details how RBAC for openHAB users is implemented in openHAB. The technical concepts of openHAB such as the architecture of openHAB with a technical perspective, the client-server model, and the access control management of the IoT platform will be outlined. The steps to build and run an instance of openHAB with changes will be specified. How roles and groups are persistently managed and how resources sent to a user are filtered based on their roles and role groups will be exposed. Implemented functions and modifications to the main UI are performed and explained to manage the RBAC model in the openHAB IoT ecosystem. By enabling RBAC for openHAB users, fine-grained access control of resources for smart family home

users (which is the illustrated scenario) is possible. Incorporating the RBAC model is not an easy task, so how the challenges encountered are overcome will be described. Finally, potential security vulnerabilities will be listed and how these vulnerabilities can be fixed with features already present in openHAB and resolved by implementing the RBAC model in openHAB.

The thesis will end with a conclusion explaining how important the RBAC model can be in openHAB.

Chapter 2

Methodology

2.1 Context

The XXI century is a generation where technologies are present everywhere and keeps on growing. The history of wireless has evolved in an extremely short time and continues its expansion. More and more devices become smart like lamps, alarms, thermostats, cameras, etc by having the ability to connect to the Internet. These new appliances are called the Internet of Things (IoT). The Internet connects individuals to the network for communication with other individuals, while the main objective of the Internet of Things is to enable objects, smart devices, or systems to be connected, interoperate, and communicate with other objects, smart devices, systems, or individuals at any time, anywhere and by using any network, path or service through the Internet as common platform [35]. IoT technology is widely used with the concept of the *smart home* [86]. A smart home is a home that contains IoT devices in its environment where the IoT devices communicate, are manageable, and programmable. A smart home is the perfect example of the omnipresence, ubiquity, and pervasiveness of computing inside a home. There are several synonyms for the smart home such as home automation, domotic, intelligent home, home care, and aware house. Lutolf defined a smart home which is as follows [43]: "*the smart home concept is the integration of different services within a home by using a common communication system. It assures an economic, secure, and comfortable operation of the home and includes a high degree of intelligent functionality and flexibility.*" A home needs three elements to be smart, firstly the home has an internal network, secondly, the home has a smart control with a gateway to manage the functionality of the system, and finally, the home can be automated [72].

The interaction of IoT devices in a smart home can also be referred as the IoT ecosystem. The IoT ecosystem in a smart home includes all the components that allow the home's resident to connect their IoT devices. The ecosystem includes remote access, dashboards, networks, gateways, analytics, data storage, and security [23]. IoT ecosystems in the context of home care are permanently connected to the Internet. Therefore that creates many security challenges because being constantly connected to the internet results in being the target of cyberattacks. The challenges of the smart home are that the privacy of the smart home occupant must be preserved and the service provided must not be subject to interference, intrusion, or obstruction. To ensure the security of a smart home, Lazakidou A., Siassiakos K., and Ioannou K. [41] enumerate six properties that must be

maintained: Confidentiality, Integrity, Authentication, Authorization, Non-repudiation, and Availability.

- **Confidentiality** to preserve unauthorized access to certain information to prevent someone from spying on the house to find out, for example, when the house alarm is turned off.
- **Integrity** to prevent an unauthorized person from tampering with sensitive data such as user credentials.
- **Authentication** is the verification of an entity using passwords or secret keys shared between two parties. This ensures that only the occupant of the home can access the home automation IoT ecosystem.
- **Authorization** determines the user's access right, as not all users have the same access right to the home automation.
- **Non-repudiation** prevents denial of the involvement action in the IoT ecosystem, such as denying having disabled the alarm or the camera.
- **Availability** ensures that the services provided in-home care are always available and are protected from events impacting the smart home, such as malicious attacks [41].

OpenHAB is the chosen platform for managing the IoT ecosystem of home automation. It already has strong security and maintains the six properties described above.

2.2 Open Home Automation Bus

Open Home Automation Bus (openHAB) created by Kai Kreuzer is an open-source, technology-agnostic home automation platform. It operates at the center of a smart home. It empowers the smart home by connecting (smart) devices and services from different vendors, including other home automation systems and technologies, integrating physical hardware, external system, and web services, involving a large number of manufacturers and subsystems [56]. All the features, components, services, devices, systems, and subsystems managed, maintained, and regulated by openHAB are handled by a uniform user interface and eventually by automation rules [60].

More technically, openHAB is written in Java and uses OSGi. OSGi is a powerful technology for the development of platforms that provides a vendor-independent, standards-based approach to dynamically modulate Java software applications and infrastructure [70]. Therefore OSGi allows the integration of almost an infinite number of add-ons. For the moment the Internet of Things (IoT) development platform offered by Kai Kreuzer has 370 add-ons, 2854 things [56] and approximately 315 bindings in his stable (3.2.0) version [66] and many more in the latest (3.3.0) version [67]. These add-ons, things, and bindings allow the connection and integration of smart devices and technologies into openHAB like Amazone Echo, Philips Hue lights, Smart Plug Wi-Fi, etc. OSGi is not the only powerful technology used by the smart home platform. There are also integrated Eclipse projects such as Equinox which serves the runtime environment [22] or Jetty which

provides a web server [36] and the modern polymorphic application container Apache's Karaf powered by OSGi, defined as a modulith runtime running on the cloud and offers features such as shell consoles, remote access, hot deployment, dynamic configuration, etc [38].

OpenHAB is an open-source project. It has many contributors partly due to Black Duck Open Hub [11] having indexed openHAB. Black Duck Open Hub is a huge team for platform development and more generally open-source software development that contains 499,107 open source projects, 5,877,971 open source contributors, and 31,541,298,095 lines of code [1]. Moreover, there is a community [57] on the openHAB website with highly active persons, who respond really quickly, where we can ask any questions related to the platform.

It is a good decision to choose openHAB to make your home smart because he has won numerous challenges and been nominated by various communities. The technology-agnostic home automation platform won the JAX Innovation Award 2014 [14] and was the People's Choice Winner at the Postscapes IoT Awards 2014/15 [7]. openHAB is also one of 11 organizations recognized by Oracle and the Java community in 2013, therefore he won the Duke's Choice Award of 2013 [85]. In addition to all these prizes, Eclipse has also confidence in the development of the openHAB project because since 2013 openHAB has been part of the Eclipse Foundation [20]. Since the platform for IoT devices became a member of the Eclipse Foundation, it benefits from many advantages [21] such as participation in a thriving developer community to increase the number of people who improve the platform or sharing of costs and innovation while focusing on creating new and differentiated technologies that the customer values most to find out which new binding (add-on) should be implemented in the platform [21].

2.3 Why should we use openHAB instead of proprietary IoT platforms?

In recent years, the number of smart devices and connected homes has increased drastically. The Internet of Things (IoT) market is one of the fastest-growing markets in the telecommunications sector [19]. Recent analyses have estimated the market for IoT devices to be worth 384.5 billion dollars in 2021 and will reach between $600 [44] and $1800 [25] billion in 2027-2018 with about 40 billion connected devices in the world [9]. Therefore, open-source IoT platforms like openHAB will have an ever-increasing demand with ever-increasing consumers. Hence the following question should be asked: why it will be better, for a consumer who wants to connect his house with smart devices, to opt for openHAB which is an open-source IoT platform instead of a proprietary IoT platform? Is it better to use openHAB rather than an IoT platform with a closed IoT ecosystem like SmarthThings, Amazon Web Services IoT, Google Cloud IoT, etc? In this section, the advantages of openHAB over closed IoT platforms will be explained and the reasons why openHAB should be preferred to manage the IoT ecosystem will be seen.

Proprietary IoT platforms have a close IoT ecosystem whereas, with open IoT platforms like openHAB, the user has full control of the IoT ecosystem. A closed IoT ecosystem

means that the user using proprietary IoT platforms does not have full control over their data, cannot modify, enhance or manage the source code, and cannot access all of the resources of the home automation software. For example, with the Google Cloud IoT platform all user data is collected, received, and centralized in a remote cloud, then combined, analyzed, and transferred for storage or publication [19]. Storing user data in a remote cloud controlled by a company brings with it the risk that the company has the possibility to use these data for the wrong purposes. Remember the Cambridge Analytica data scandal [82] where Facebook users' data was used without their consent for political advertising purposes. This is still a risk of using proprietary IoT platforms. For openHAB this risk is non-existent because only the user will have access to their data, so no company will process their data and no one will be able to access this data unless the user has been hacked.

Closed IoT platforms like Google Cloud IoT process the data generated by smart devices in a smart home remotely in a cloud. Thus, these data must be sent remotely to a cloud, which leads to more security issues as user data flows across multiple networks. OpenHAB's philosophy is completely different. All data from IoT devices is handled only on the local network, the Intranet of Things should rather be said than the Internet of Things. Consequently, managing data only on the local network greatly reduces the probability of being hacked [47, 42].

A newly created application can be easily incorporated into openHAB because there are fewer permission restrictions and less risk, which leads to innovation. For example, if a startup, company, or even individual creates a new IoT device, the Binding to connect that new IoT device to openHAB can be directly implemented, then tested, and used by many openHAB users. OpenHAB can provide an opportunity for a company, start-up, or individuals to do innovation and creation when this is not possible with a closed IoT platform because the source code is not accessible and can not be modified [39].

With openHAB it is possible to stay on top of the technology and ensure good security by using different open-source tools and technologies. In fact, for this thesis, the openHABian open source IoT framework has been combined with a Raspberry Pi. OpenHABian installs, configures, maintains, and runs openHAB on a Raspberry Pi. A Raspberry Pi is open-source hardware that acts as a proxy for the openHAB platform. TailScale VPN [79] has been used to set up a VPN between the raspberry Pi and the computer. The VPN encrypts all data exchange between the computer and the Raspberry Pi. Through the VPN the Raspberry Pi can be reached remotely by using SSH[1] and allows to connect to the openHAB platform. This example shows that it is possible with openHAB to connect remotely to his smart home while keeping strong security. All the while knowing that the data remains and are processed on the local network in the Raspberry Pi. Only the data requested by the user (when it is outside the local network) are sent outside the local network. Moreover, the data sent outside the network is encrypted by the VPN. In the case of a closed IoT platform, all data are sent and processed remotely.

The openHAB user interface is highly customizable while for the closed IoT platforms

[1]SSH allows to register and execute commands securely in a remote machine [26]

it is limited. In openHAB 3 [60], it is even possible to automatically create pages with extensive use of the semantic model. Indeed for SmartThings which is a closed IoT platform the dashboard cannot be managed in a browser [18]. With openHAB, the user interface can be fully managed in a browser with widgets, sitemaps, text files, etc.

OpenHAB is free while proprietary IoT platforms are highly expensive. In fact, Google Cloud IoT [30] charges monthly based on the volume of data used, or with AWS IoT on their website a scenario is explained where the monthly fee is high at $22.59 [2]. It depends on what the consumer wants, but these prices are really high compared to the free open source automation software openHAB.

OpenHAB offers the ability to provide gateways between different software, hardware, and systems to communicate with each other and allows the IoT platform to be interoperable. Indeed, openHAB can combine Samsung SmartThings Binding, Google Assistant Action, Amazon Echo Control Binding, and many others. An interesting example of how useful openHAB's interoperability is compared to enterprise IoT platforms that essentially accept their own smart devices is with SmartThings. SmarthThings closed IoT platform does not support Google Chromecast [24] while openHAB [56] can integrate SmartThings devices and Google Chromecast. This example clearly demonstrates the interoperability of openHAB which can handle IoT devices provided by different companies.

In conclusion, it is an excellent choice to use openHAB rather than IoT platforms with a closed ecosystem. In the next section, we will discuss why openHAB is a well-rated open-source IoT platform.

2.4 Evaluation of open-source IoT platforms for smart homes

The demand for smart devices is increasing at a high speed. Therefore to manage this increasing number of IoT devices many open source automation software exists to centralize and handle these smart devices for a domotic. Is openHAB really a good open-source IoT platform compared to other existing open-source IoT platforms? This question is answered in the article of Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. [76] where a complete evaluation of the four best existing systems (chosen according to some criteria) will be performed and a final score will be assigned to these four selected systems.

Initially, Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. [76] compiled a list of twenty existing open-source home automation software. This list of twenty systems is composed of consulting search engines such as Google, Wikipedia, Bing, and online Internet hosting providers for software development and version control such as Github, Gitlab, etc. Once the list is compiled by Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. [76], a selection of five systems from the list of twenty systems was made in order to reduce the number of systems subject to further analysis. The selection was made by Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. [76] by ranking each system from the list of twenty systems found based on four criteria. The criteria defined by Setz

B., Graef S., Ivanova D., Tiessen A., and Aiello M. [76] are the number of commits, the number of stars for the source code repository, the last commit, and the quality of the documentation, and the number of contributors. Each criterion has a score between zero and five. The criteria scores for each system are added together by Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. [76] to get the final score. The four open-source home automation systems with the highest score in the list of twenty are Home Assistant, Domoticz, openHAB, and ioBroker.

The in-depth analysis realized by Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. [76] of the four selected open-source home automation systems consists of a use case-based analysis and a criteria-based analysis. The features for the use case-based analysis were extracted by Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. [76] from 17 use cases divided into five categories. The five categories of features for the use of case-based analysis are visualization, location, notification, data processing, and interaction. A detailed explanation of each feature for each category of the use of case-based analysis is explained by Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. in their article [76]. For use case-based analysis, Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. [76] assign a score to each of the four open-source home automation platforms based on how well the features are handled for each of the four systems. Home Assistant scores 4.5, Domoticz scores 2, openHAB scores 4, and ioBrocker scores 3.

Based on each feature of the use case-based analysis, Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. evaluate 34 criteria, divided into 8 categories which are popularity and community, price, configuration complexity, user interface, and user experience, security and authentication, scalability and support, system performance, and software quality. The detailed evaluation of each criterion and the different scores assigned to each criterion for each of the four selected systems are explained in detail by Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. in their article [76]. Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. calculate the final score assigned to each of the four open-source home automation platforms by adding up all the feature and criteria scores. The result obtained by Setz B., Graef S., Ivanova D., Tiessen A., and Aiello M. [76] is as follows: first place goes to Home Assistant with a score of 3.9, second place goes to openHAB with a score of 3.7, third place goes to ioBrocker with the score of 3.4 and fourth place goes to Domoticz with the score of 3.1. OpenHAB finishes in second place, therefore it is a famous, powerful, and widely used open-source home automation software. The conclusion is that openHAB is a good open-source IoT platform compared to other existing open-source IoT platforms.

2.5 Motivation

OpenHAB is a large, well-known, and robust IoT platform with highly active users, thus security is a fundamental aspect that cannot be neglected. Access control is one of the main mechanisms to ensure good security within an IoT platform such as openHAB. Therefore, what is the current access control for openHAB users? In this section, the current access control for openHAB users will be shown. Then, a scenario will be illustrated to show why openHAB's access control is limited [33].

2.5.1 Actual access control

At the moment, there are only two types of users in openHAB, the *administrator* and the *user*. The administrator has all the rights. He or she can manage the rules to automate the smart home, customize the user interface with widgets and pages, and configure the smart home. By default, when a user is created, he becomes the user of type *user* and can access all resources of the main interface. Thus, when a user has access to the main interface of the platform, all IoT devices in the smart home ecosystem are accessible to them. However, the user can only manipulate the available buttons and see the information displayed on the main interface. He cannot change the smart home configuration, rules, and pages. In openHAB, the two user types *administrator* and *user* are defined as roles. Therefore, access control can be supervised only with these two roles and cannot be customized for each user.

Creating or deleting a user can be done with the openHAB console, it is not possible with the main interface. The commands available to manage a user are shown in Figure 2.1.

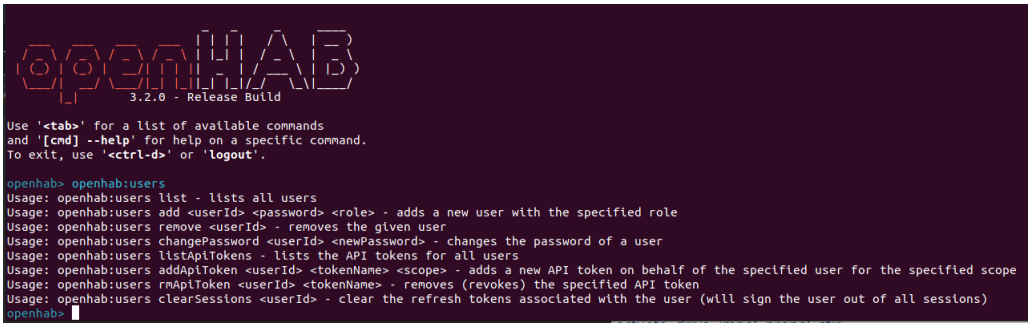

Figure 2.1: Available commands for users management (openHAB console)

It is possible to restrict access to devices, functionalities, or sources of information in the main UI to all users or no users. For example for a created layout displayed in Figure 2.2, it is only possible to manage the visibility of the role *administrator* or the role *user*.

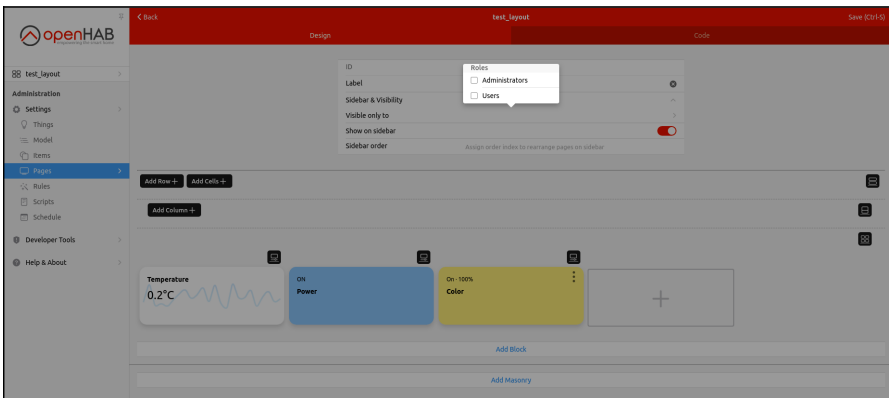

Figure 2.2: Visibility management of a layout (main UI)

The administrator has the ability to prevent a button on the main UI from being clicked

by allowing only read access. The disadvantage is that everyone, even administrators, will not be able to click on that button on the main UI until an administrator allows write access for that button again.

By default, operations requiring the role *user*, such as access to the main UI, layouts, sitemaps, etc., are available even if they are not authenticated. It is possible to force authentication for *user* role operations by disabling the button in Settings → API Security → Enable/Disable Implicit User Role.

In conclusion, access control for openHAB users is managed for two types of users: the *administrator* and the *user*. The platform components are displayed to all users or to no users. Therefore openHAB has almost no access control for its users.

2.5.2 Scenario

Now a scenario will be introduced to illustrate why openHAB user access control needs to be improved, and why it is very limited, and not flexible at all.

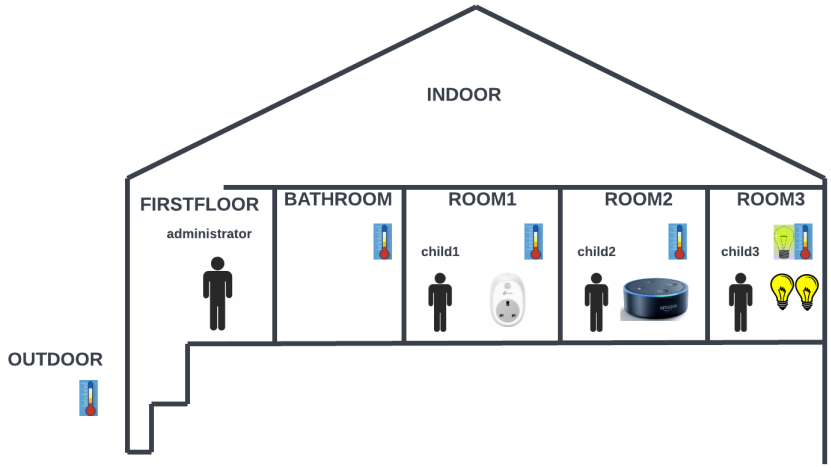

Figure 2.3: Scenario

The scenario is a smart house with a family as shown in Figure 2.3. The father of the family will have the role of *administrator* as he is the head of the family. The father of the family will configure and install the smart devices in the smart home for his three children named *child1*, *child2*, and *child3*. Children have the role of *user*s to prevent the children from changing the rules or configurations of the smart home. In the displayed scenario in Figure 2.3, there are temperature sensors outside and inside (each room) of the home automation system. In Room1, there is a Wi-Fi plug connected to one lamp to turn it on and off. In Room2 there is Amazon Echo where Alexa is installed. Amazon Echo is a device that can be controlled by voice and has many features. In Room3 there are two white Philips Hue lamps, which are simple white lamps, and one colored Philips Hue lamp, which is a lamp that can be changed in color.

In the scenario, the father is the *administrator*. Therefore, the father is able to create rules such as turning off the lights in Room3 at 10 pm, starting the Amazon Echo music at 8 pm every day except Saturday and Sunday to wake up the child2 in Room2 and for Room1 he can turn off the Wi-Fi plug at 10 pm every day to turn off the light. For the temperature, if an indoor temperature exceeds a certain limit like 20 °C, the father will be alerted by notification and he will turn off the heater of the appropriate rooms.

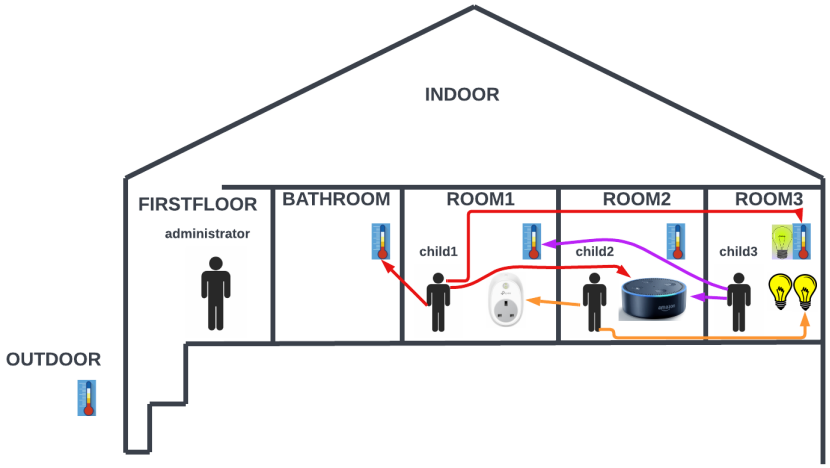

Figure 2.4: Scenario without user access restrictions

OpenHAB is already well built to automate the smart home with rules. However, if a child wants to access their own devices in their room, they will also have access to all other devices in the smart home. A child has the role of the *user* so either all children have access to a device or no child has access to a device. Therefore, this is a problem, the Figure 2.4 illustrates the limited access control management of openHAB as each child can manage his own devices and also those of his brothers. For example, child2 can turn on the light of the child3 of Room3 very late at night to annoy his brother or child1 can start the Amazon Echo music in Room2 very late at night to wake his brother. Having each child have the ability to manage the devices in another room will cause litigation between the children and be very annoying to them.

In conclusion, openHAB is capable of allowing access to smart home devices to all users or no users. There is no way to customize access control for every user, location, and device in the openHAB IoT ecosystem.

2.6 Access control models

OpenHAB has almost no access control for its users. Therefore, an access control model must be incorporated. There are many access control models for managing access to users and resources of a platform, such as discretionary access control, mandatory access control, role-based access control, or attribute-based access control. Which of these access control models should be incorporated into openHAB? Which access control models will be most appropriate for openHAB's requirements for access control of its users? In this section, for

each of the four access control models, a description will be made as to why the access control model will or will not be implemented in openHAB [40, 33].

2.6.1 Discretionary Access Control (DAC)

The Discretion Access Control model gives a user full control over a resource. The user can create, modify, or manipulate the object if he or she is authorized to do so. This model analyzes who can access which object with an authorization database that contains all authorized users. It also manages permission inheritance and audits system events and administrative privileges. It is generally used to manage file access, for example, it is used by the UNIX family and the Windows operating system. It is very flexible because the owner can customize the access policies and permissions of the created object for each user [40, 16]. The DAC model is illustrated in Figure 2.5 (Langaliya C., Aluvalu R., February 7, 2015).

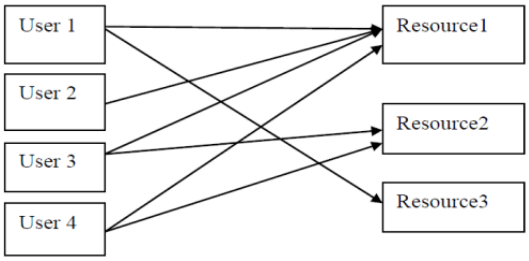

Figure 2.5: Discretionary access control (Langaliya C., Aluvalu R., 7 February 2015) [40]

DAC is already in place in openHAB to handle file access control because openHAB is built and runs on an operating system that is Linux, Windows, or macOS where the DAC model is used. Therefore, there is no need to incorporate this model into openHAB because it already exists.

2.6.2 Mandatory access control (MAC)

Mandatory access control is provided by a central authority that manages the security policy for the entire system. Permission and access to an object are not managed by the object owner but by the authority. No one other than the authority can override the security policy, permissions, and access. The authority defines which user can access which resources. Therefore, the data owner cannot manage information from a high privacy object to a low privacy object. In addition, a user cannot access the data if they do not have the authorization to manage it [40]. MAC is used for the confidentiality of a system. It is mainly adopted for military or governmental security where the access of the data owner is decided by the authority [16]. This model is illustrated in 2.6 (Langaliya C., Aluvalu R., 7 February 2015).

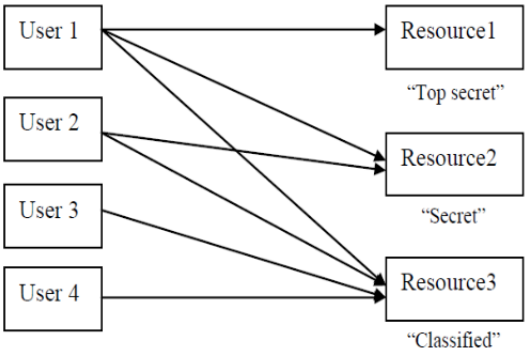

Figure 2.6: Mandatory access control (Langaliya C., Aluvalu R., 7 February 2015) [40]

This model can be useful for openHAB because the integrity and sensitive data flow of the smart home will be highly protected. However, with MAC, once a user reaches a certain security level, that security level can no longer be changed. For example, if a user who has had access to top-secret level data can no longer access a lower secret level of data, the privacy of the data may be compromised. Thus, this model leads to a restriction that can be very annoying for the smart home occupant. Therefore, the MAC will not be incorporated into the openHAB [16].

2.6.3 Role-based access control (RBAC)

Role-based access control model assigns roles to users. A role is attributed to a user defining the resources he/she can access as well as the resulting responsibilities from that access. All users' access is treated with roles. Based on these roles, the system identifies the roles belonging to a user to know the user's access. In RBAC, a hierarchical structure of roles is defined to manage the security of the system, where for example the role *administrator* has all access and privileges and thus users with this role define roles and assign roles to other users [40]. RBAC allows complex controls such as segregation of duties to be implemented. This model is illustrated below 2.7 (Langaliya C., Aluvalu R., 7 February 2015).

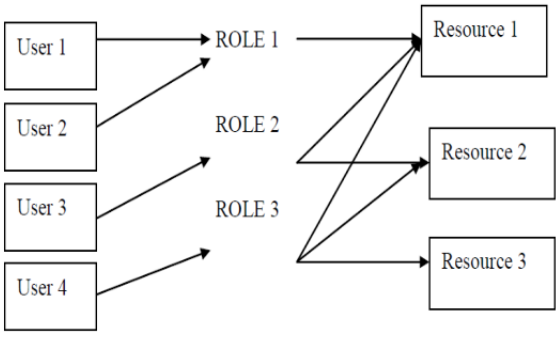

Figure 2.7: Role-based access control (Langaliya C., Aluvalu R., 7 February 2015) [40]

RBAC is widely used for web servers and web service platforms to assign privileges to roles. As stated in the NIST Cybersecurity Framework [77], web services platforms such as

openHAB must be implemented at least for the administrator, developers, and any other privileged accounts needed for the web services platform to function properly. This is the case with openHAB where there are two roles: the *administrator* role and the *user* role.

RBAC will be the chosen access control model for openHAB because it meets well the requirements of the missing functionalities for access control management of the scenario explained in the Section 2.5.2. Indeed, with this model, the least privilege required to perform an action can be assigned to a role such as the smart home location(s) or device(s) [77]. Thus, by taking into account the scenario, the father will be able to designate roles for his children to give them the minimum permissions necessary to access and manipulate their devices. There will be no litigation between the children because each child will have access to his own devices and will not have access to devices that are not authorized to him.

However, it is necessary to be careful with the RBAC model because the privileges given to a role, the roles assigned to a user, and the creation of roles are performed by a user with the role *administrator*. Therefore, some conflicts are possible in the definition of roles and their access, which may lead to security problems.

2.6.4 Attribute-based access control (ABAC)

The attribute-based access control model is realized with identification, authentication, authorization, and accountability [40]. In ABAC, each user has attributes to access system resources. The set of attributes considered by the ABAC model for the resource requested by a user depends on the identity and characteristics of the user, the type of resource the user wishes to access, such as a web service, system function, or data, and the operational, technical, or situational environment or context in which access to the resource occurs [77]. ABAC is highly adjustable, flexible, and dynamic because the attributes requested for a resource depend on three elements: the subject, the resource, and the environment [40, 77]. The Figure 2.8 (Ce Hu V., Ferraiolo D., Kuhn R., Schnitzer A., Sandlin K., Miller R., and Scarfone K., January 2014) represents how ABAC works. Where object attributes are equivalent to resource attributes [15].

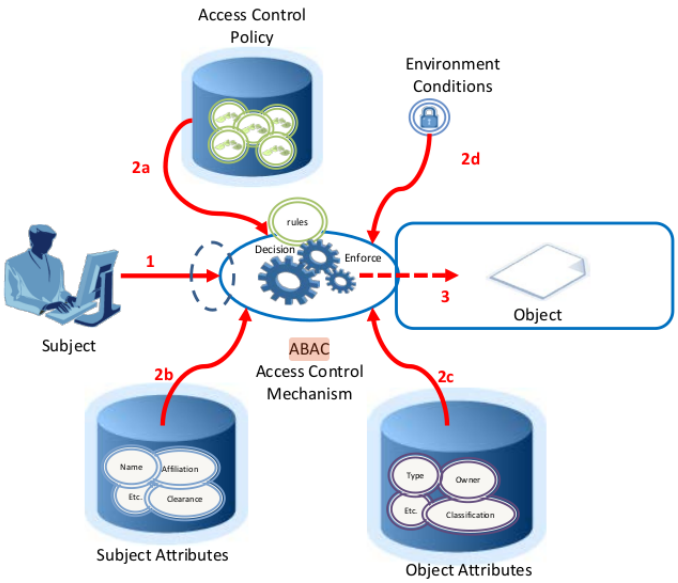

Figure 2.8: Attribute-based access control (Ce Hu V., Ferraiolo D., Kuhn R., Schnitzer A., Sandlin K., Miller R., and Scarfone K., January 2014) [15]

The explanation of the steps in the Figure 2.8 is explained by Ce Hu V., Ferraiolo D., Kuhn R., Schnitzer A., Sandlin K., Miller R., and Scarfone K. [15] and is as follows:

1. *Subject requests access to the object.*
2. *Access Control Mechanism evaluates:*
   (a) *Rules*
   (b) *Subject Attributes*
   (c) *Object Attributes*
   (d) *Environment Conditions to compute a decision*
3. *Subject is given access to an object if the object is authorized.*

The ABAC model is more flexible, secure, and fine-grained than the RBAC model because, first, it takes into account the platform environment. Second, policy rules can be customized and defined with the semantic context in mind. And third, the management and adjustment of access to a resource are done with a set of attributes instead of a single role [15]. However, this model will not be implemented in openHAB because ABAC does not give the concepts of a role assigned to the user that openHAB is already based on [40]. ABAC could be implemented in openHAB because it provides a high level of security, but this requires a lot of management inside the platforms, and a huge amount of work and to have such a level of security is not necessarily required by openHAB because openHAB (as said in Section 2.3) manages IoT devices in the local network and not remotely.

2.7 Objective

The role-based access control model is chosen for incorporation into openHAB because it is the most appropriate model for managing access control for openHAB users given the

scenario described in Section 2.5.2. Incorporating RBAC into openHAB is not an easy task and requires a lot of platform changes. Access control is one of the key components of openHAB and handles the most important parts and features of the home automation software. Therefore, the core of the source code project openHAB needs to be modified, adjusted, and updated. In this section, the RBAC model embedded in openHAB will be defined and the different objectives that need to be achieved to properly manage the roles and access to the platform resources will be explained.

The RBAC model implemented in openHAB manages roles and groups of roles. Each group is composed of multiple roles and each role is composed of multiple unique identifiers that identify a platform resource. Groups are also managed by the RBAC model incorporated in openHAB to easily add/remove multiple roles to users. Thus, when a user logs in to the main home automation software interface, openHAB with the RBAC model identifies the roles and role groups assigned to the user. Based on these roles and role groups, openHAB will obtain a set of unique IDs. Each unique ID represents a resource of the IoT ecosystem. With this set of unique IDs, openHAB will know which resources will be allowed for the user.

In order to integrate the RBAC model into openHAB, several roles need to be taken into account in openHAB because currently there are only two roles that are managed by the platform. It will be possible to add, delete or modify a role or group stored on the platform. Adding, deleting, and modifying a role or group to a user will also be feasible. A role must define the resources to which access is allowed for users who contain the role. Groups must contain several roles. A user should therefore only have access to the platform's resources according to his roles and groups. Finally, only users with the role of *administrator* will be able to manage the RBAC model and users in openHAB.

Now that the RBAC model incorporated in openHAB and the objectives that must be achieved to integrate it properly have been described, in the next section, an explanation of the contribution implemented to openHAB will be made.

2.8 Contribution

Integrating RBAC into openHAB is a complex task that needs to be taken seriously. For the thesis, what are the contributions made to openHAB? The different repositories of openHAB will be presented and how to propose a change to one of these repositories will be done. Then, what has been added to openHAB will be explained. Finally, the contribution realized to the openHAB community will be detailed.

The home automation software is set out with 44 repositories available on Github [61]. The four main projects are the openhab-core, openhab-addons, openhab-distro and openhab-webui. The **openhab-core** repository is the core implementation of the platform, **openhab-addons** contains all the Add-ons (such as bindings, voice services, integration systems, automation systems, data persistence services, and data transformation services), **openhab-distro** contains all the parts for assembling the binary distribution and **openhab-webui** is the official web user interface of openHAB. To implement the role-based

access control for openHAB users only the openhab-core project and the openhab-webui project will be modified. The openhab-distro repository will be used to test the openHAB source code changes locally.

To improve, revise and update these 44 repositories and implement the role-based access control model for openHAB users, contribution guidelines are explained in the openHAB documentation [58]. The proposals for change (the pull request action to submit some modification to a repository on Github) must be accepted by the administrators of openHAB who have the main access, some static code analysis has to be enforced and coding guidelines have to be respected [59]. A pull request [27] has been made to solve an issue [50] that asked to add/remove user roles via the Karaf console. The full code implementing role-based access control for openHAB users is available on a fork made to the openhab-core project at the branch temp and on a fork to the openhab-webui project [61].

As shown in the Figure 2.9 (openHAB documentation, updated 22 December 2020) [60] in the documentation of the home automation software, it is possible to manage openHAB in three different ways. Either with the text files, with the main UI, or with the openHAB console which is the Karaf console. For time reasons it will not be possible to manage roles and role groups with text files, but the beginning of the interpreter has been done. For the main UI, also for time reasons, the complete management of the role-based access control model has not been done. However, the users and their corresponding groups and roles as well as the available groups and roles are displayed on the main UI. In addition, it is possible to create a group or a role and save it on openHAB. Finally, with the openHAB console, it is possible to completely manage the role-based access control model for openHAB users. All commands to manage groups and roles are available and implemented in the Karaf console.

Configuration Task	via text files	in Main UI	openHAB console	Recommendation
Auto-Discover Things and Items	✕	✓	✓	Main UI Do not autocreate Items
Define and manage Things	✓	✓	✓	Main UI
Define and manage Groups and Items	✓	✓	(✓)	for starters: Use the semantic model in UI advanced users: import items in UI to use the semantic model only stick with *.items files if you know how to handle tagging and groups for the model
Define GUI	sitemaps only	✓ includes YAML view	✓	Main UI sitemaps/*.sitemap files
Define Transformations	✓	✕	✕	transform/*.map*.js files
Define Persistence	✓	✕	✕	persistence/*.persist files
Define Rules	✓	✓	✕	for starters: Main UI and Blockly (graphically create JS code) for advanced users: rules/*.rules files for rules DSL and JSR223
Manage Z-Wave Devices	✕	✓	✕	Main UI
Modify openHAB Settings/Services	✓	✓	✓	Main UI
Install Add-ons	✓	✓	✓	Main UI

Figure 2.9: Comparison between configurations (openHAB documentation, updated 22 December 2020) [60]

Specifically, the functionalities added to openHAB are listed below:

- Roles and groups are stored permanently in a file.
- List available roles and groups.
- Add, remove, change a role or a group stored in openHAB.
- Add, remove, change a unique ID that identifies a resource of the platform to a role.
- List available roles and groups for a user.
- Add, remove, change a role or a group to a user.
- Only users with the administrator role can add or remove a user.
- Only users with the administrator role can add, remove or change a role or a group to a user.
- Authenticate the user for each request made to the platform in order to know which resources must be made available to him for each of his requests.
- The validity and the integrity are checked for each request to avoid usurped, altered, or tempered requests and to deny unauthorized access to resources.
- Available roles and groups are displayed in the main UI.
- Available user roles and groups are displayed in the main UI.
- Create a role or a group with the main UI and store it permanently.
- Display to the main UI the resources, data, and functionalities of the domotic authorized to the user according to his roles and groups.

To develop these functionalities a lot of problems, issues and challenges have been encountered. To correct, resolve and overcome these complications, questions have been posted to the openHAB community. All available questions asked and the summary of the activity to integrate RBAC into openHAB is described in the openHAB community [69]. A total of 20 topics and 37 posts (a post is an answer to a topic) have been created. The openHAB community has been visited 93 days and 257 topics and 1 800 posts have been viewed with a total read time of 22 hours. The activity in this community demonstrates the difficulty of incorporating the RBAC model into openHAB and the complexity of the platform.

Chapter 3

Overview of openHAB

This chapter first describes openHAB's main concepts and architecture. Following this, a detailed description of the configuration of the scenario (see Section 2.5.2) will be depicted. Finally, the chapter explains where the configuration files must be put in openHABian. This will allow us to understand how to manage and access openHABian, a version of openHAB for a Raspberry Pi.

3.1 Concepts

To get a general understanding of how to set up IoT devices in his smart home with openHAB, the main concepts of openHAB will be introduced [60].

Bindings are software packages that can be installed by the users with the role *administrator* and are provided in the add-on section. A Binding makes several Things available. Its purpose is to make a connection between the device and the Thing. The Binding interacts directly with the device and sends all the messages, information, and commands to the Thing. The interaction between the device and the Thing is with real devices in addition to simulated ones, web services, or any other manageable source of information and functionality.

Things are software, made available from a Binding, representing a device, a functionality, or a source of information. It is the physical layer as shown in Figure 3.2. The Things make their functionalities available with Channels. Multiple devices can be connected to one Thing. The Thing is mostly used for the configuration process and can have configuration properties such as an IP address or an access token. The Thing can be seen from the user's point of view as a set-up but cannot be used for the operations such as managing the behavior of a device.

Items are the representation of specific devices, locations, equipment, or properties and their respective information. It is the virtual layer as shown in Figure 3.2 openHAB separates the application into Things (which represent the real world) and Items (which represent the virtual world). Items represent the functionality used by the application, create events through the event bus in an asynchronous way, can be managed with the events, and must be defined according to the semantic model. The semantic model will be

explained further.

Channels are provided by Things and make the connection between Things and Items. Channels allow to create a **Link** between the Things and the Items and define how they communicate with each other. Channel may be linked to multiple Items and Item may be linked to multiple Channels.

The five concepts: **Bindings**, **Things**, **Channels**, **Items** and **Links** are used to connect devices to the openHAB platform and to configure the scenario, in this case a family house.

The Figure 3.1 (openHAB documentation, updated 22 December 2020) [60] represents the five concepts. In the Figure 3.1, there is an actuator, which is one of the Things available on the previously installed Binding. The actuator has a physical address and must be configured in order to be used. On the right of the Figure 3.1, two Items can be found representing two lights in the user interface of the platform. There are two Channels to make the link between the Thing and the Items allowing the user to control the lights with the user interface by turning on and off a button. The buttons create events on the event bus hence managing the behavior of the lights. Only two Channels are used by the Thing (in the case of the Figure 3.1 the Thing represents an actuator). Therefore the Thing is not used at its full capacity.

The Figure 3.2 shows how the five concepts of openHAB work together. In the Figure 3.2, one Channel can be linked to two Items, an Item can be linked to two Channels coming from the same or different Things.

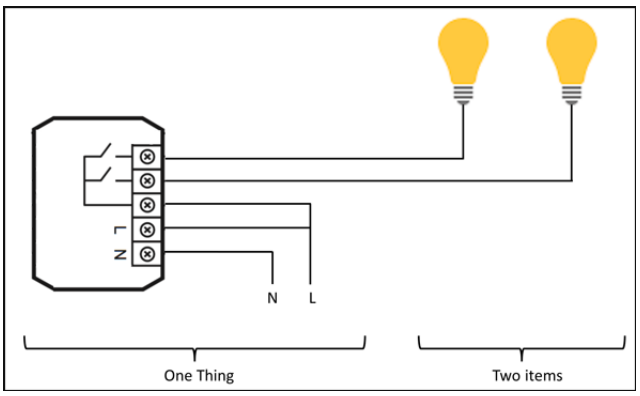

Figure 3.1: openHAB documentation - concept overview (openHAB documentation, updated 22 December 2020) [60]

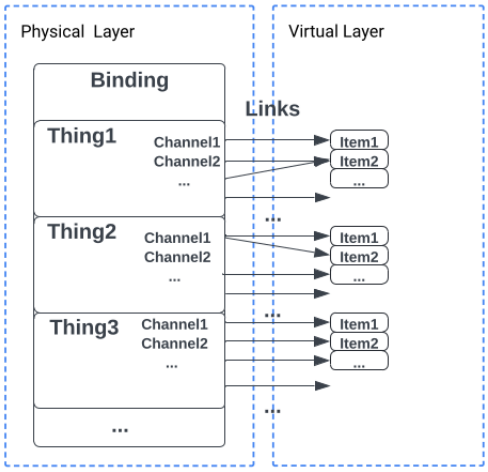

Figure 3.2: Concepts of openHAB

3.2 openHAB architecture

In the Figure 3.3 (Heimgaertner F., Hettich S., Kohlbacher O. and Menth M., 2017) openHAB is shown as a whole. On top, in the first layer, there is the automation part, the

Items, and the different UIs. The basic UI is a web interface based on Google's Material Design Lite that does not rely on any JavaScript framework [31]. Generally, interface graphics of platforms are based on a javascript framework. The classic UI is quite identical to the basic UI, but the interface is a little different. The paper UI is the one helping with the configuration in openHAB. The rest API [60] is the part of openHAB for exchanging information between the server-side and the client-side with the HTTP protocol. The rest API provides on the main UI the data related to the five concepts (Binding, Thing, Channel, Item, Think) and the resulting actions of these concepts to change the behavior of the smart home IoT ecosystem configured with openHAB. The main UI includes the basic UI, the classic UI, the paper UI, and the rest API [60]. In the second layer (starting at the top) there is the event bus to control openHAB. In the third layer (starting at the top), there are the Bindings that interact using several different protocols with different services, hardware, or information sources. Finally, the fourth layer (below the Figure 3.3) represents the devices and network services of the smart home.

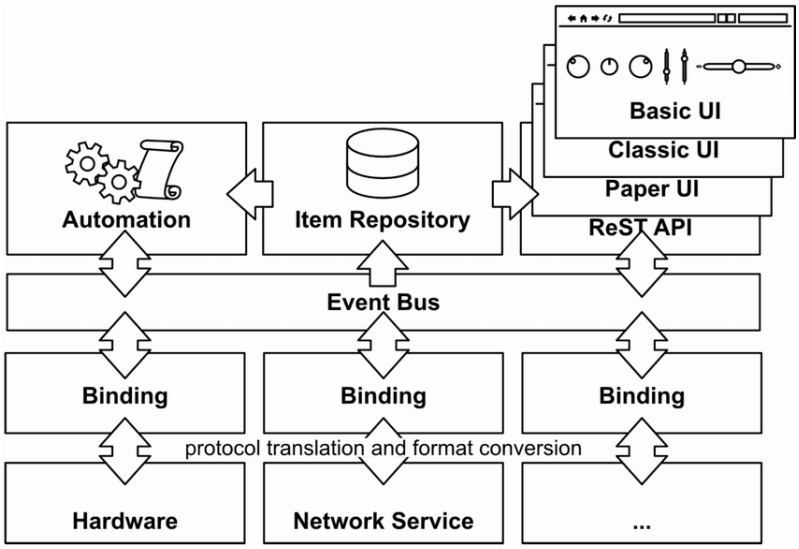

Figure 3.3: openHAB architecture (Heimgaertner F., Hettich S., Kohlbacher O. and Menth M., 2017) [34]

3.3 The configured scenario

OpenHAB can be installed with a Raspberry Pi, Linux (with two different package managers, *.deb* or *.rpm*), Windows, macOS, or Docker with three different versions (Stable, Milestone, and Snapshot). The smart family house, in our scenario, was configured with a Raspberry Pi that uses openHABian [62]. openHABian auto-configures the Linux operating system and installs openHAB on the Raspberry Pi. OpenHABian can be installed on the Raspberry Pi by following the steps explained in the openHAB documentation [51]. The Raspberry Pi acts as a proxy for the smart home, allowing the openHAB platform to run continuously and always be available. A VPN has been set up on the raspberry PI with Tailscale VPN [79]. Tailscale VPN is one of the openHABian [62] features rendering it possible to access the smart home with any Wi-Fi (not only with local Wi-Fi) so almost everywhere (even abroad). The implemented scenario is a house with a family which

represents well the usefulness of the RBAC model for openHAB users as each user wants to access its own devices in its room and the devices in the common space should be available for everyone.

As explained in Section 2.9, a scenario can be configured with text files, with the main UI, or with the openHAB console. For our scenario (see Section 2.5.2), the text files will be used for the representation of the smart home and the definition of the Items simulating temperatures. The main UI will be used to add new Items. For the configuration, the openHAB console will not be used because it is less user-friendly than the text files and the main UI.

Firstly, the syntax description will be done to understand how to define an item in a text file. Secondly, the smart home configuration will be explained.

3.3.1 Item syntax

To define an Item in a text file, the following syntax must be respected [60]:

```
itemtype itemname "labeltext [stateformat]" <iconname> (group1, group2, ...)
["tag1", "tag2", ...]  {binding configuration}
```

`itemtype` and `itemname` are mandatory, all other fields are optional [60].

Itemtype defines the state that can be stored by the Item and the commands it will accept. The available Item types are shown in Figure 3.4 (openHAB documentation, updated 22 December 2020) [60].

Type Name	Description	Command Types
Color	Color information (RGB)	OnOff, IncreaseDecrease, Percent, HSB
Contact	Status of contacts, e.g. door/window contacts. Does not accept commands, only status updates.	OpenClosed
DateTime	Stores date and time	-
Dimmer	Percentage value for dimmers	OnOff, IncreaseDecrease, Percent
Group	Item to nest other items / collect them in groups	-
Image	Binary data of an image	-
Location	GPS coordinates	Point
Number	Values in number format	Decimal
Player	Allows control of players (e.g. audio players)	PlayPause, NextPrevious, RewindFastforward
Rollershutter	Roller shutter Item, typically used for blinds	UpDown, StopMove, Percent
String	Stores texts	String
Switch	Switch Item, used for anything that needs to be switched ON and OFF	OnOff

Figure 3.4: Available Itemtypes, openHAB documentation (openHAB documentation, updated 22 December 2020) [60]

ItemName defines the unique name attributed to the Item to identify it.

Label is for the state representation of the Item.

Icons allows selecting the image to be displayed on the user interface next to the item name.

Groups specify to which group the Item belongs.

Tag specifies the nature of the Item. If it is a Location, a Point, a Property, or Equipment.

Binding configuration looks like `<bidingId>:<thingId>:<uniqueRandomNumber>:<channelId>`. **Binding configuration** is the combination of the Binding ID, the Thing ID that is one of the Things of the specified Binding, and the Channel ID that is one of the Channels of the specified Thing. Sometimes there is more information such as in the case where the same Thing comes up several times. The Binding configuration is used to link the Item to a Thing.

3.3.2 Semantic model

As explained earlier in Section 3.1, the Items are the representation of devices, functionalities, or information sources of the smart home configured with openHAB. To properly represent the smart family home, we need to organize the Items. Items can be configured together in various ways. The attention will focus on the semantic model to correctly assemble the Items and get a correct representation of the scenario in openHAB which is a smart family house.

The semantic model is divided into four main concepts; Location, Point, Property, and Equipment. The relationship between these concepts is shown in Figure 3.5 (openHAB documentation, updated 22 December 2020) [60] and has to be respected.

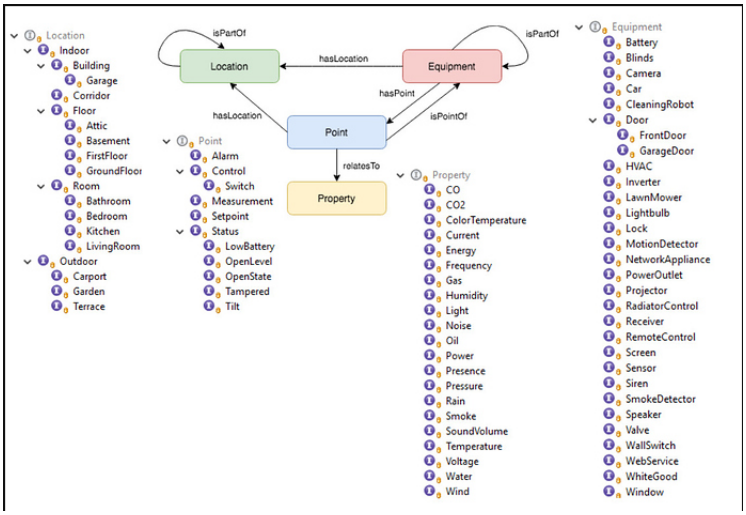

Figure 3.5: openHAB documentation - Semantic model (openHAB documentation, updated 22 December 2020) [60]

The relationships between the concepts are [60]:

- A **location** is an Item with the Item type *Group* (displayed in 3.4). It includes sublocations, equipment, and points. It represents the physical location of the smart home, such as a room, bathroom, interior, exterior, etc.
- An **Equipment** is an Item with the Item type *Group*. It includes Sub-Equipment and Points.
- A **Point** is represented by all the Item types except the Item type *Group*. It is usually linked to a Channel.
- A **Property** is additional information about an Item of the type Point that indicates what type of Point it is. For example, a thermometer might be a Point of type Measurement with a Property of type Temperature [60].

In addition to these relationships there are restrictions listed below, collected from [60]:

- A **Location** must be the member of a Location or the member of nothing.
- An **Equipment** must be the member of a Location, Equipment, or the member of nothing.
- A **Point** must be the member of a Location, Equipment, or the member of nothing.

3.3.3 Installed Bindings and Things

This section explains which Bindings and Things were installed for the scenario that is a house with a family (see Section 2.5.2) and how the Things in each Binding were configured.

1. **Amazon Echo Control Binding** for Amazon Echo where Alexa is installed.

   → Amazon Echo has been connected to the local network with the phone app Alexa.

   (a) Alexa application has been installed on the phone with either App Store, Google Play or Android.

   (b) The connection to our Amazon account has been made or an account has been created if there is none.

   (c) The application will discover the Amazon Echo device on the local network (your local WIFI), the connection to the Amazon Echo device has been made to the local network with the Amazon Alexa application.

   (d) For the scenario, Spotify has been configured for Alexa, so Alexa has direct access to Spotify and it will be possible to launch Spotify with the voice recognition of Amazon Echo or with openHAB.

   (e) The steps to configure Spotify on the Alexa Application is explained on the website of Spotify [78].

   → The configured Things for the Amazon Echo Control Binding are the followings:

(a) The Thing **Amazon Echo** requires the serial number of the Amazon Echo device.

- The serial number has been obtained by putting the following link on a browser:
  - `http://openhab:8080/amazonechocontrol/`
  - `openhab` is the IP address where openHAB is hosted, the port 8080 is because by default openHAB is running on this port.
  - For openHAB hosted on the Raspberry Pi, the IP address is the IP address generated by Tailscale VPN [79] and the port is 8080 because openHAB listens on this port.
- Then, The connection to the Amazon account must be made.
- Finally the Figure 3.6 is displayed, where the serial number in the red frame has been recovered.

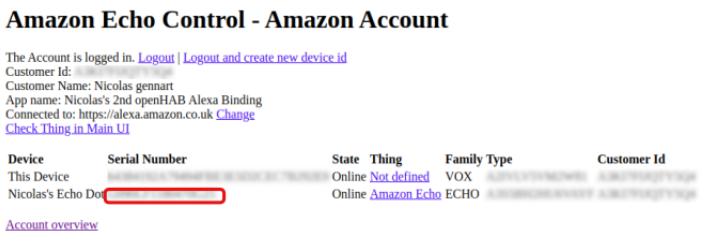

Figure 3.6: Serial number of Amazon Echo

(b) The Thing **Amazon Account** does not require any configuration because it is already done when the connection to the Amazon account has been established to obtain the serial number of the Amazon Echo device.

2. **D-Link SmartHome Binding** for the camera. Unfortunately, this Binding only allows the Thing for the camera DCH-S150 and the camera is type DCS-8000LH. So it was not possible to add this type of camera to the smart home with openHAB because the thing is missing.

3. **TP-Link Smart Home Binding** for the Wi-Fi Plug.

   → The Wi-Fi Plug has been connected to the local network with the mobile app Kasa that can be downloaded with App Store, Google Play or Android.

   → The Thing installed and configured to connect the Wi-Fi Plug to openHAB is the **HS110 Smart Wi-Fi Plug**. This Thing is automatically discovered by openHAB once connected to the local network with the Kasa application and can be added.

4. **Philips Hue Binding** for the two Philips Hue With lamps and the Philips Hue color lamp.

   → To discover the Philips Hue lamps in the local network, the bridge was connected to the Wi-Fi box with an Ethernet cable. Once connected all the Things that

must be installed to connect the two Philips Hue lamps and the Philips color lamp are displayed by openHAB, then were installed and are as follows: two Things type **hue:0100** for the two Philips Hue with lamps and one Thing type **hue:0210** for the Philips Hue color lamp.

5. **Nibe Heatpump Binding** to simulate temperatures.

   → The Thing installed for this Binding is the **nibeheatpump:f1x45-simulator** and requires no configuration.

The Figure 3.7 shows all the things that are online.

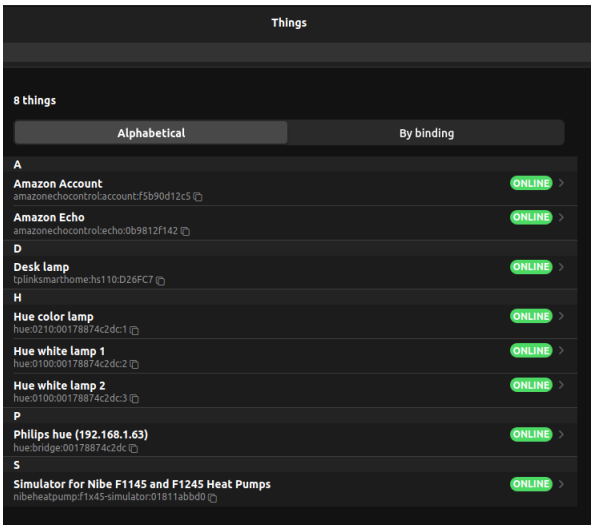

Figure 3.7: Available online Things

Now that the configuration of Items with text files has been explained. How the Bindings and Things have been installed and configured was explained and the description of the scenario has been done. The description of how the scenario has been built will be done.

3.3.4 Configuration

The part where the configuration of Items has been made with a text file.

The house will be separated into two parts: Indoor, and Outdoor. The Item for the Indoor is shown in Figure 3.8 and the Item for the Outdoor is shown in Figure 3.9. As explained in Section 3.3.2, an Item representing a Location must have the Group Item type so the Items gIndoor and gOutdoor have the Group Item type.

```
//INDOOR
Group gIndoor "Indoor" ["Indoor"]
```

Figure 3.8: Indoor Location

```
//OUTDOOR
Group gOutdoor "Outdoor" <terrace> ["Outdoor"]
```

Figure 3.9: Outdoor Location

In the Outdoor Location there is an outdoor temperature simulator. The link between the Item and the Thing is made and defined with the ID of the Thing **nibeheatpump:f1x45-simulator:27dc8d91dc** and with the ID of the Channel **sensor:#40004** as described in the binding page and shown in the Figure 3.10 [60].

```
Number nOutdoorTemperature "Temperature [%.1f°C]" <temperature> (gOutdoor) ["Measurement", "Temperature"]
    {channel="nibeheatpump:f1x45-simulator:27dc8d91dc:sensor#40004"}
```

Figure 3.10: Outdoor temperature

In the Indoor location there is the First Floor location and it contains the Items gRoom1, gRoom2, gRoom3, and gBathroom which represent locations. In the Figure 3.11, the Items gRoom1, gRoom2, gRoom3, and gBathroom have as a parent the Item gFirstFloor and the Item gFirstFloor has as a parent the gIndoor.

The Figure 3.12 represents the Equipments of the first floor, they are not Locations because they contain the Tag **Sensor** which is an Equipment as explained in the Section 3.3.2 and displayed in the Figure 3.5 with the relationship between concepts.

```
//FIRST FLOOR
Group gFirstFloor "First Floor" <firstfloor> (gIndoor) ["FirstFloor"]

Group gRoom1 "Room1" <bedroom> (gFirstFloor) ["Room"]

Group gRoom2 "Room2" <bedroom> (gFirstFloor) ["Room"]

Group gRoom3 "Room3" <bedroom> (gFirstFloor) ["Room"]

Group gBathroom "Bathroom" <bath> (gFirstFloor) ["Bathroom"]
```

Figure 3.11: Locations of the first floor

```
// Equipements
Group gBathroomSensor "Bathroom Sensor" <bath> (gBathroom) ["Sensor"]

Group gRoom1Sensor "Room1 Sensor" <bath> (gRoom1) ["Sensor"]

Group gRoom2Sensor "Room2 Sensor" <bath> (gRoom2) ["Sensor"]

Group gRoom3Sensor "Room3 Sensor" <bath> (gRoom3) ["Sensor"]
```

Figure 3.12: Equipments of the first floor.

Each room and the bathroom have an indoor temperature simulator defined in the same way as the outdoor temperature. The only difference is the Channel ID as it is an indoor temperature simulator. The Channel ID is **sensor#40007** as described in the binding page [60]. These Items are Points and Properties as they contain the Tags **Measurement** and **Temperature** as defined in the semantic model Figure 3.5.

```
//Points and Properties
Number nBathroomTemperature "Temperature [%.1f°C]" <temperature> (gBathroomSensor) ["Measurement", "Temperature"]
    { channel="nibeheatpump:f1x45-simulator:27dc8d91dc:sensor#40007"}

Number nRoom1Temperature "Temperature [%.1f°C]" <temperature> (gRoom1Sensor) ["Measurement", "Temperature"]
    { channel="nibeheatpump:f1x45-simulator:27dc8d91dc:sensor#40007"}

Number nRoom2Temperature "Temperature [%.1f°C]" <temperature> (gRoom2Sensor) ["Measurement", "Temperature"]
    { channel="nibeheatpump:f1x45-simulator:27dc8d91dc:sensor#40007"}

Number nRoom3Temperature "Temperature [%.1f°C]" <temperature> (gRoom3Sensor) ["Measurement", "Temperature"]
    { channel="nibeheatpump:f1x45-simulator:27dc8d91dc:sensor#40007"}
```

Figure 3.13: Indoor temperature

The global configuration file is shown in the Figure 3.14 and the scenario is displayed in the Figure 3.15 where all simulated temperatures are displayed at each location.

```
// the scenario
//INDOOR
Group gIndoor "Indoor" ["Indoor"]


//FIRST FLOOR
Group gFirstFloor "First Floor" <firstfloor> (gIndoor) ["FirstFloor"]

Group gRoom1 "Room1" <bedroom> (gFirstFloor) ["Room"]

Group gRoom2 "Room2" <bedroom> (gFirstFloor) ["Room"]

Group gRoom3 "Room3" <bedroom> (gFirstFloor) ["Room"]

Group gBathroom "Bathroom" <bath> (gFirstFloor) ["Bathroom"]

// Equipements
Group gBathroomSensor "Bathroom Sensor" <bath> (gBathroom) ["Sensor"]

Group gRoom1Sensor "Room1 Sensor" <bath> (gRoom1) ["Sensor"]

Group gRoom2Sensor "Room2 Sensor" <bath> (gRoom2) ["Sensor"]

Group gRoom3Sensor "Room3 Sensor" <bath> (gRoom3) ["Sensor"]

//Points and Properties
Number nBathroomTemperature "Temperature [%.1f°C]" <temperature> (gBathroomSensor) ["Measurement", "Temperature"]
    { channel="nibeheatpump:f1x45-simulator:27dc8d91dc:sensor#40007"}

Number nRoom1Temperature "Temperature [%.1f°C]" <temperature> (gRoom1Sensor) ["Measurement", "Temperature"]
    { channel="nibeheatpump:f1x45-simulator:27dc8d91dc:sensor#40007"}

Number nRoom2Temperature "Temperature [%.1f°C]" <temperature> (gRoom2Sensor) ["Measurement", "Temperature"]
    { channel="nibeheatpump:f1x45-simulator:27dc8d91dc:sensor#40007"}

Number nRoom3Temperature "Temperature [%.1f°C]" <temperature> (gRoom3Sensor) ["Measurement", "Temperature"]
    { channel="nibeheatpump:f1x45-simulator:27dc8d91dc:sensor#40007"}

//OUTDOOR
Group gOutdoor "Outdoor" <terrace> ["Outdoor"]

Number nOutdoorTemperature "Temperature [%.1f°C]" <temperature> (gOutdoor) ["Measurement", "Temperature"]
    {channel="nibeheatpump:f1x45-simulator:27dc8d91dc:sensor#40004"}
```

Figure 3.14: Scenario with only simulated temperatures.

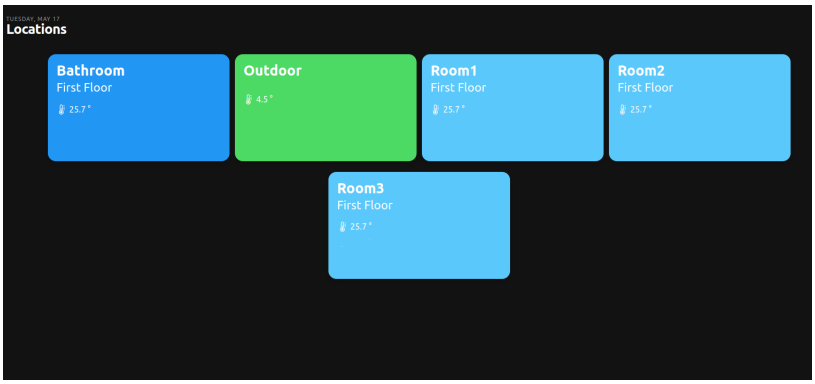

Figure 3.15: Global view of the scenario.

The part where the configuration of Items has been made with the main UI.

This part can also be configured with the text file.

How to add one Item and link this Item to a Thing through a Channel with the main UI will be shown with the **AmazonEcho_Player** Item to stop and play music. The other Items are added in the same way. For other Items, the unique name of the Item will be specified. The configuration Binding to know how the Item is linked to the Thing and, through which Channel will be stated. Finally, the Group the Item belongs to and the added metadata (if there are any images that will be shown) to the Item will be detailed. Other features will be added to the Item automatically by the main UI, such as Item type,

icons, label, etc (see Section 3.3.1). So they will not be described in the explanation.

Once the Thing Amazon Echo is found in the main UI, the tab *Channel* in the red frame of the Figure 3.16 has been clicked.

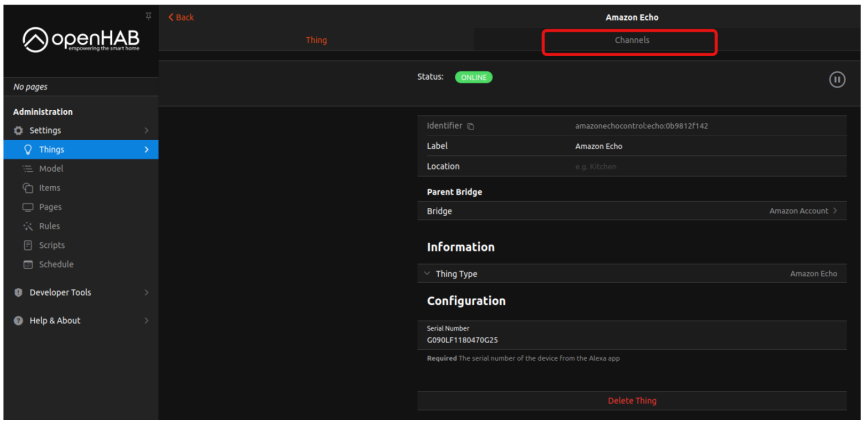

Figure 3.16: Amazon Echo Thing

Next, all available Channels were displayed for the Amazon Echo Thing. Linking an Item to the Channel Player with the ID **player** was done by clicking on the *Add Link to Item...* button in the red frame of the Figure 3.17.

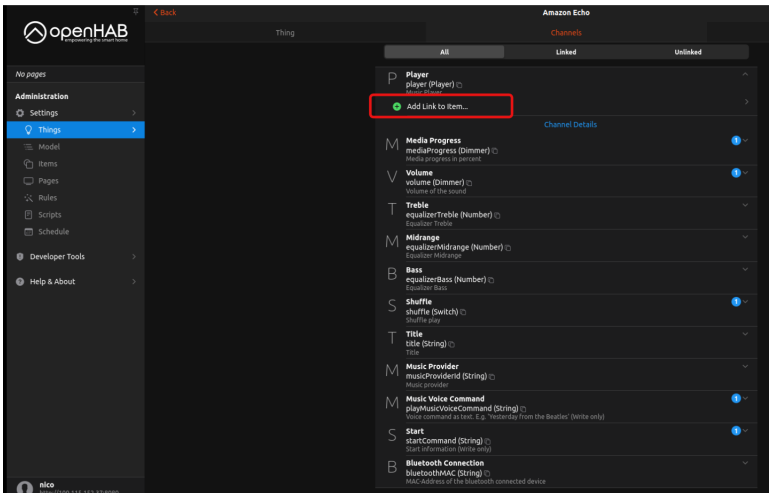

Figure 3.17: Available channels for the Thing Amazon Echo

Then, the *Create New Item* button was clicked as shown in the red frame of the Figure 3.18 and the groups were chosen by clicking on the *Parent Group(s)* button in the second red frame of the Figure 3.18. The parent group chosen for the **AmazonEcho_Player** Item was gRoonSensor2 as shown in the Figure 3.19. The blue *Link* button in Figure 3.18 was clicked to finalize the configuration of the **AmazonEcho_Player** Item. The created **AmazonEcho_Player** Item for the Channel Player can be seen as shown in Figure 3.20.

The final result of this configuration will give as Binding configuration **amazonechocontrol:echo:<uniqueRandomNumber>:player** where **amazonechocontrol** is the ID of the Amazon Echo Binding, **echo** is the ID of the Amazon Echo Thing and **player** is the ID of the Player Channel. (see Section 3.3.1)

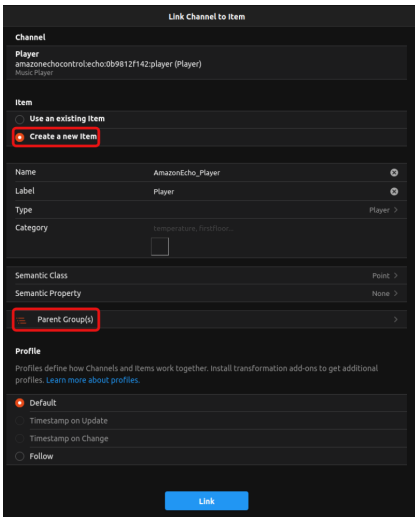

Figure 3.18: Creation of an Item

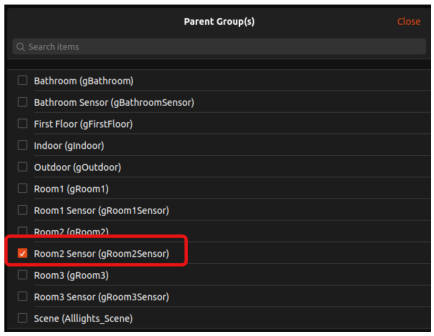

Figure 3.19: Parent group for the Item **AmazonEcho_Player**

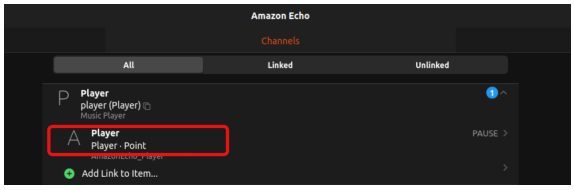

Figure 3.20: Liaison of the **AmazonEcho_Player** Item to the Channel **player**.

Now the explanation of the other created Items will be done.

Configured Items for the HS110 Smart Wi-Fi Plug Thing

The Thing HS110 Smart Wi-Fi Plug with the ID **hs110** and the Binding configuration **tplinksmarthome:hs110:<uniqueRandomNumber>:switch** has the Item **Desklamp_Power** links to the Channel Power with the ID **switch** that allows to turn on and off the Smart Wi-Fi Plug. His direct parent is the Group **gRoom1Sensor**, thus the Item is displayed in the virtual room one on the user interface as shown in Figure 3.21

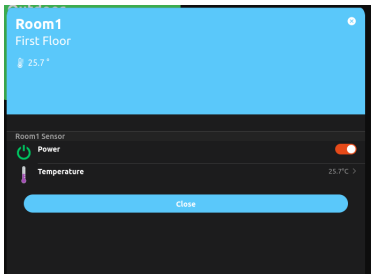

Figure 3.21: Virtual room one on the main UI

Configured Items for the Amazon Echo Thing

For Amazon Echo Thing with the ID **echo** and the configuration Binding **amazonechocontrol:echo:<uniqueRandomNumber>:<channelId>** (the different channel IDs are listed below). For this Thing, in addition to the Item **AmazonEcho_Player** (already explained above in Section 3.3.4), there are the following Items:

- **AmazonEcho_MediaProgress** to see the progress of the music and allows you to advance or rewind the music link to the Channel Media Progress with the ID **mediaProgress**.
- **AmazonEcho_Volume** to see the current volume of the music link to the Channel Volume with the ID **volume**.
- **AmazonEcho_Shuffle** to shuffle the next play music link to the Channel Shuffle with the ID **shuffle**.
- **AmazonEcho_MusicVoiceCommand** to launch Spotify link to the Channel Music Voice Command with the ID **playMusicVoiceCommand**. For this Item, metadata has been added to launch Spotify on Amazon Echo device as displayed in Figure 3.22.

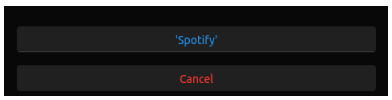

Figure 3.22: Command for Spotify on Amazon Echo device.

The metadata is added for the Item **AmazonEcho_MusicVoiceCommand** by following the Figure 3.24, Figure 3.25 and then the Figure 3.26.

- **AmazonEcho_Start** to execute some strings to Alexa (installed in Amazon Echo). **AmazonEcho_Start** Item is linked to the Channel Start with the ID **startCommand**. For this Item, metadata has been added to send the String command to the Amazon Echo device as shown in Figure 3.23.

Figure 3.23: String commands.

The metadata is added for the Item **AmazonEcho_Start** in the same way as the added metadata of the Item **AmazonEcho_MusicVoiceCommand** except that the command options added are different. The command options for the Item **AmazonEcho_Start** are displayed in Figure 3.27.

All the Items of the Amazon Echo Thing have been added to the gRoom2Sensor Group so all these Items are available in virtual room two on the user interface as shown in the Figure 3.28.

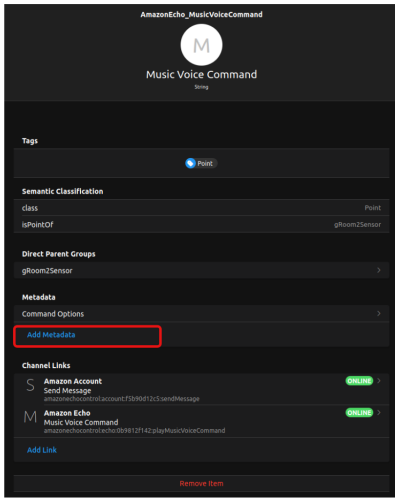

Figure 3.24: New metadata on an Item

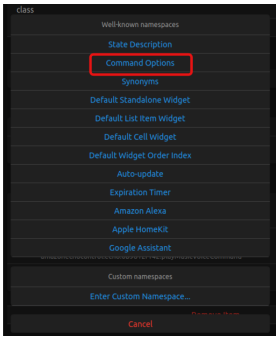

Figure 3.25: New command options for an Item

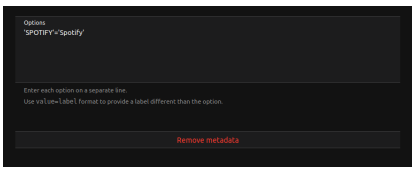

Figure 3.26: New command Spotify

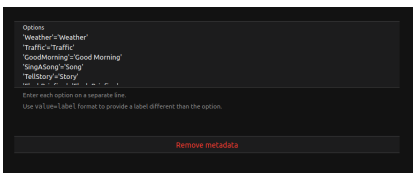

Figure 3.27: New Strings commands

Configured Items for the Amazon Account Thing

The Amazon Account Thing with the ID **account** and Binding configuration **amazonechocontrol:account:<uniqueRandomNumber>:<channelId>** has only

one Channel which is Send Message with the ID **sendMessage**. This Channel allows sending a String command to Alexa (in Amazon Echo) instead of a voice command. The Items link to the Channel **sendMessage** are the Items **AmazonEcho_MusicVoiceCommand** and **AmazonEcho_Start**. These Items have already been created in the Amazon Echo Thing. The channel **sendMessage** must be set up to allow the Item to send a String command to Alexa. Therefore, the **AmazonEcho_MusicVoiceCommand** Item and the **AmazonEcho_Start** Item both have two Channels (as explained in the Section 3.1 an Item can have two channels). The Item **AmazonEcho_MusicVoiceCommand** has the Channels **sendMessage** and **playMusicVoiceCommand** and the Item **AmazonEcho_Start** has the Channels **sendMessage** and **startCommand**. Both Items have the gRoom2Sensor Group as their parent, so they are displayed in virtual room two on the user interface as explained in the Section 3.28.

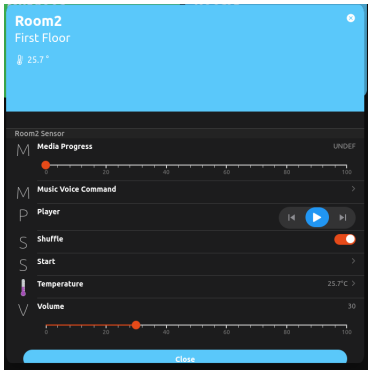

Figure 3.28: Virtual room two on the main UI

Configured Items for the Thing type 0210 from the Philips Hue Binding

The Thing of type 0210 with the Binding configuration **hue:0210:<uniqueRandomNumber>:1:<channelID>** has the Item **Huecolorlamp_Color** link to the Color Channel with ID **color** to change the color of the Philips Hue Color lamp and the Item **Huecolorlamp_ColorTemperature** link to the channel **color_temperature** to change the color temperature of the Philips Hue Color lamp. Both of these Items have a parent of **gRoom2Sensor** and are therefore displayed in the virtual room three on the user interface as shown in Figure 3.29.

Configured Items for the Thing type 0100 from the Philips Hue Binding

The Thing of type 0100 with the Binding configuration **hue:0100:<uniqueRandomNumber>:2:<channelID>** has the Item **Huewhitelamp1_BEB** link to the Brightness Channel with the ID **brightness** to change the brightness of the Philips Hue White lamp. There are two Philips Hue White lamps and therefore another Thing with the configuration Binding **hue:0100:<uniqueRandomNumber>:3:<channelId>** and an item with the name **Huewhitelamp2_BEB** also link to the Brightness Channel with ID **brightness** to change the brightness of the Philips Hue White lamp.

The two Items **Huewhitelamp1_BEB** and **Huewhitelamp2_BEB** have as parent **gRoom3Sensor** thus they are displayed in the virtual room three on the user interface as shown in Figure 3.29.

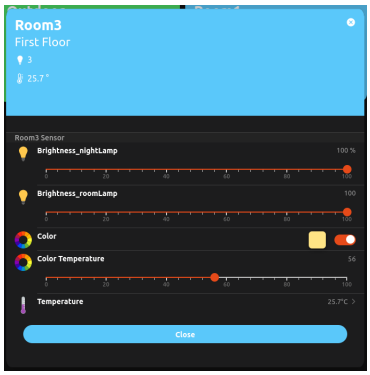

Figure 3.29: Virtual room three on the main UI

The Figure 5.1 (in the appendix) shows the global configuration of the smart home with all the Items in compliance with the semantic model (see Section 3.3.2).

3.4 Location of the configuration files and access to openHABian.

In the case of the Raspberry Pi, the configuration file has to be put at the location `/etc/openhab/items` of the file system of the Raspberry Pi. This allows openHAB to consider the configuration files of the Items displayed in Figure 3.14.

The location `/etc/openhab/items` can be reached by accessing with **SFTP**[1] or with **SSH**[2] to the file system of the Raspberry Pi where openHABian is hosted. With SFTP the command is:

```
sftp://openhabian@<tailscaleRaspberryPiIpAddress>/
```

and with SSH the command is:

```
ssh://openhabian@<tailscaleRaspberryPiIpAddress>/
```

Where `<tailscaleRaspberryPiIpAddress>` is the IP address generated by Tail Scale VPN for the Raspberry Pi to be able to reach it remotely.

These commands for Linux users should be placed where it is displayed: *Enter the server address* in the Figure 3.30 to add a location in the file system of the Linux computer.

[1] SFTP is a network protocol used for secure file transfer via a secure shell
[2] SSH allows to save and execute commands securely on a remote machine [26]

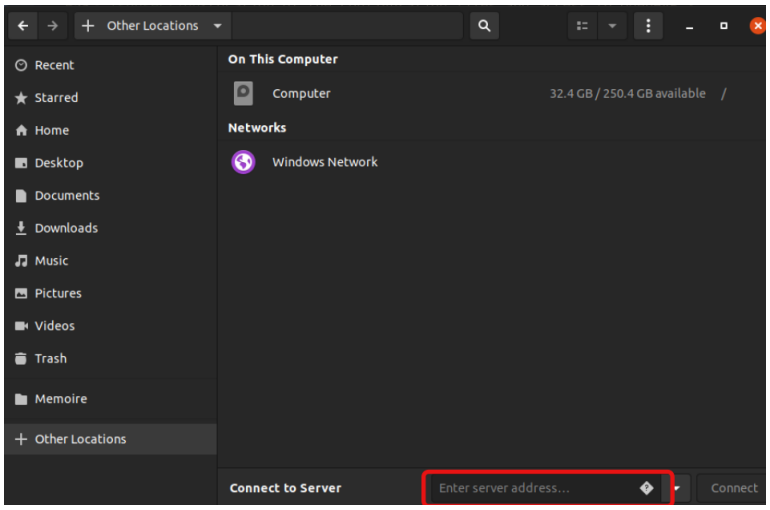

Figure 3.30: Addition of a new location on the file system of the operating system Linux

The path of `/etc/openhab/items` in the file system of the Raspberry Pi is shown below:

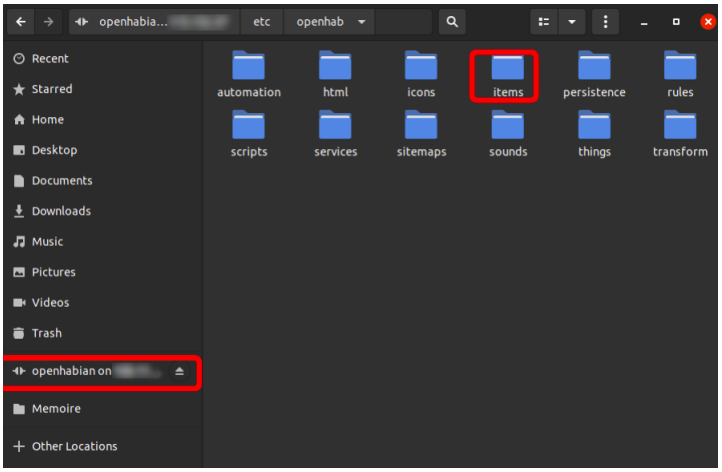

Figure 3.31: Connection to the file system of the Raspberry Pi

To access the main UI of openHAB **remotely**, hosted in the Raspberry Pi, the following url must be placed on a browser:

`http://<tailscaleRaspberryPiIpAddress>:8080`

or to access it **locally**, the following URL must be placed on a browser:

`http://openhabian:8080`

The result of the command just above is shown in the Figure 3.32.

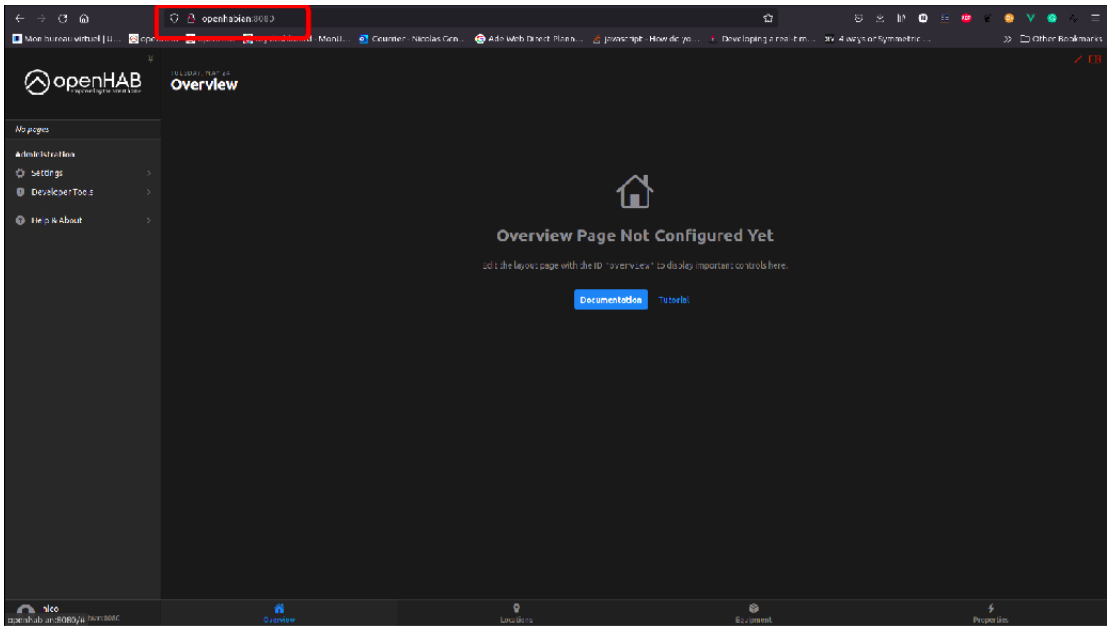

Figure 3.32: Connection to the main UI of openHABian

To access the Karaf console (locally and remotely), the following command must be executed on a terminal where the openHAB console listens on port 8101.

```
ssh -p 8101 openhab@<tailscaleRaspberryPiIpAddress>
```

The default password of the openHAB console is *habopen*. The Figure 3.33 shows what is displayed once connected to the openHAB console.

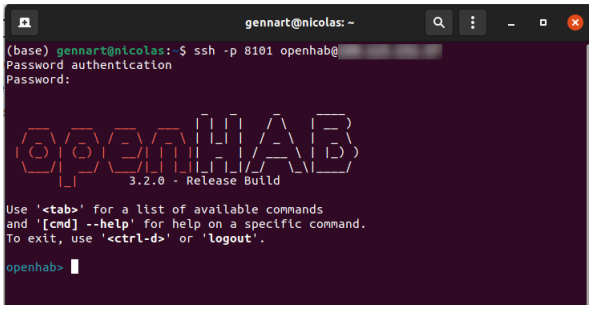

Figure 3.33: openHAB console connection

The console of the Raspberry Pi is reached (locally and remotely) by executing the following command :

```
ssh openhabian@<tailscaleRaspberryPiIpAddress>
```

The result of the command is shown below:

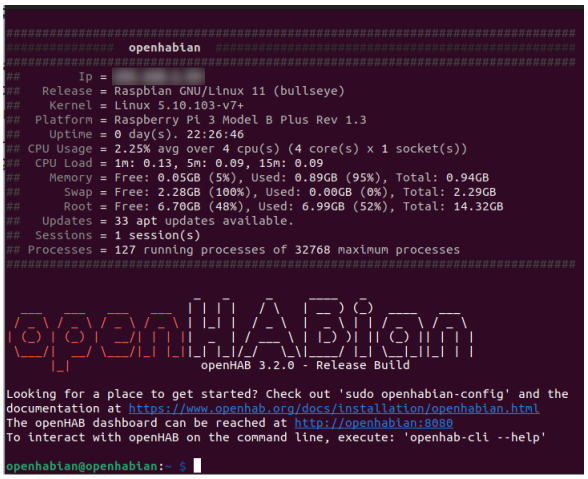

Figure 3.34: Connection to the console of the Raspberry Pi with the user openhabian

The user *openhabian* is created when openHABian is installed on the Raspberry Pi. The user *openhabian* has superuser rights on the Raspberry Pi which is the highest privilege for a user under Linux which is the operating system auto-configured when openHABian was installed on the Raspberry Pi.

Chapter 4

Implementation of RBAC for openHAB users

4.1 Technical concepts of openHAB

4.1.1 Architecture with a technical perspective

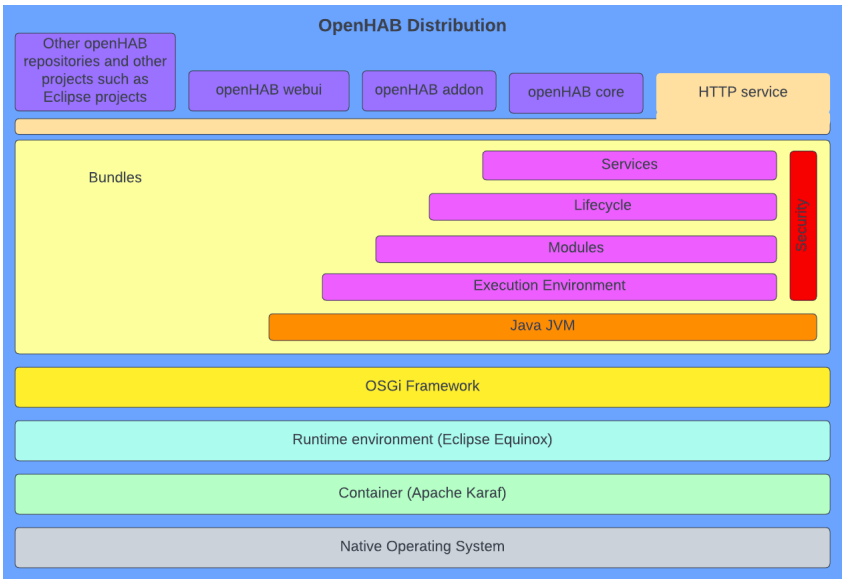

Figure 4.1: openHAB architecture from a technical perspective [12, 60]

Figure 4.1 shows the architecture of openHAB from a technical perspective. The **first layer** is the native operating system which can be Windows, Linux, or macOS. openHAB can be hosted in a container with Docker or in a Raspberry Pi where Linux is self-configured during the openHABian installation.

The **second layer** is the Apache Karaf container which provides many features such as the shell console, remote access, dynamic configuration, etc [38]. The Karaf console allows to interact directly with the server by typing and executing commands, to add/remove a user, a group, or a role for example without interacting with the user interface. The shell

console can be reached locally or remotely with SSH[1]. It represents the openHAB console.

The **third layer** is the Eclipse Equinox runtime environment [22] for the OSGi framework.

The **fourth layer** is the OSGi framework that provides a modular architecture. Modularity allows for easy management of application complexity, parallel execution of jobs, and tolerance of uncertainty [70]. The modular architecture is composed of dynamic module systems for Java represented as bundles. Dynamic because bundles can be managed at runtime. Bundles communicate with each other through services.

Bundles are represented in the **fifth layer** of the figure 4.1. A bundle is represented by JAR[2] file which is the aggregation of Java classes and associated metadata and resources, with a .jar file extension, and is composed of the following layers [60]:

1. **Services** allow the communication between the bundles.
2. **Life Cycle** defines the state of the bundle. The Figure 4.2 (OSGi Alliance, June 2019) [71] shows the different states of a bundle and how it moves from one state to another. There are 6 states [71].
   (a) **INSTALLED**: The bundle has been installed.
   (b) **RESOLVED**: All the java classes required for the bundle are available. This state indicates that the bundle is ready to be started or to be stopped.
   (c) **STARTING**: The bundle is being started. The java class that contains the @Activate annotation from the maven repository [75] is called. When subject to a lazy activation policy, the bundle remains in the STARTING state.
   (d) **ACTIVE**: The bundle has been successfully activated. the bundle runs correctly on the platform. The method with the @Activate annotation has been called and returned.
   (e) **STOPPING**: The bundle is stopped. The bundle stops running on the platform.
   (f) **UNINSTALLED**: The bundle is uninstalled. Its state cannot be changed.

   When a bundle is installed, it remains in the persistence storage of the platform until it is explicitly uninstalled. Whether the bundle is resolved, started, or stopped while the platform is running, it remains in the persistence storage of the platform [71].
3. **Modules** determines the external dependencies and the dependencies on other bundles of openHAB. Dependencies mean that if bundle A depends on bundle B, bundle A can switch to the ACTIVE state only when bundle B has the ACTIVE state.

[1] SSH allows to save and execute commands in a secure way in a remote machine [26].

[2] *A JAR (Java ARchive) is a package file format used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file for distribution. JAR files are archive files that include a Java-specific manifest file. They are built on the ZIP format and typically have a .jar file extension [83].*

4. **Execution Environment** specifies the Java environment.

5. **Java VM** is the abbreviation for Java Virtual Machine, in the case of openHAB the JVM version is Java 11 and the recommendations are explained in the documentation of openHAB [60].

6. **Security** is optional and allows to manage the permission for the different bundles.

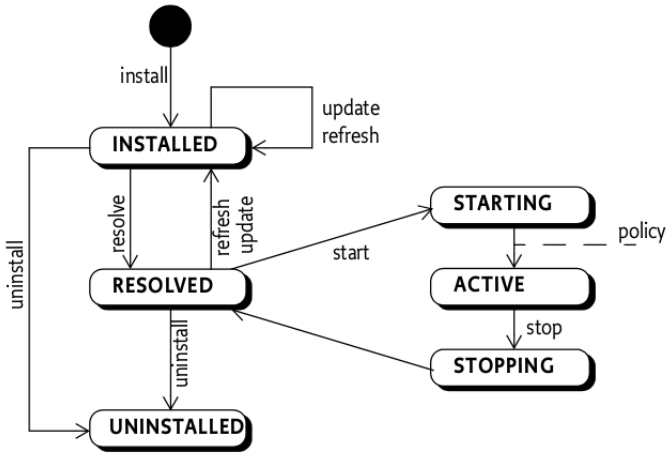

Figure 4.2: The life cycle of a bundle (OSGi Alliance, June 2019) [71]

The **last layer** at the top of the Figure 4.1 represents the services, features, repositories, and projects implemented by openHAB built and run on multiple bundles. These bundles have a *Maven* infrastructure and thus are built with the Apache Maven project [45]. OpenHAB core and openHAB webui are the modified part of openHAB to incorporate the RBAC model. OpenHAB addon is not modified but it is used to configure the scenario seen as a house with a family (see Section 2.5.2).

- **HTTP service** provided by the Eclipse Jetty project [36] includes an embedded HTTP server and allows realizing the interaction between the client-side and the server-side by executing HTTP requests/responses and by managing the HTTP[3] protocol.

- OpenHAB core represents the core functionality of openHAB, such as user management, Items management, logins, automation, etc.

- OpenHAB webui manages the user interface.

- OpenHAB addon implements all the add-ons and contains the links to connect for example the Philips Hue light, the Amazon Echo, or the TP-Link Wifi plug to the platform (see Section 3.3.4).

[3]HTTP *stands for "Hypertext Transfer Protocol." HTTP is the protocol used to transfer data over the web. It is part of the Internet protocol suite and defines commands and services used for transmitting webpage data. HTTP uses a server-client model. The client-side, for example, can be on a home computer, laptop, or mobile device. The HTTP server is typically a web host running web server software[80].*

- There are many other repositories [61] on openHAB such as openhab-google-assistant [60] which allows performing some actions in openHAB for Google assistance such as *Activate hello*, *Activate movie mode*, etc [32] [60] or the openhab-alexa to connect openHAB to Alexa and perform some actions too. Other projects such as Eclipse projects are EMF uses for the chart page [54] or Xtext to define rules and sitemaps [88].

4.1.2 Client-server model

In the next image, the model client-server is shown.

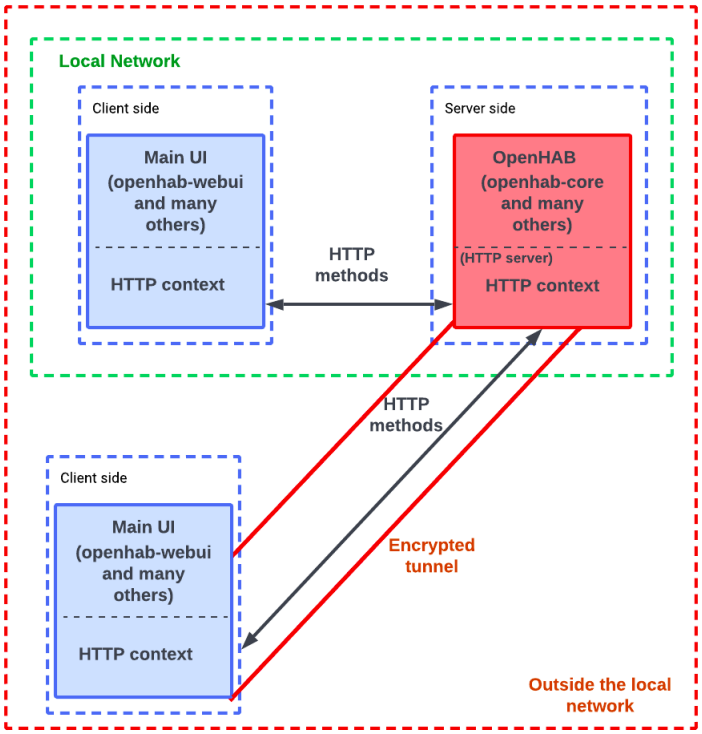

Figure 4.3: client-server model

From the Figure 4.3, the client-side can be a home computer, a laptop, or a mobile device. In the case of the computer, the client-side connects to openHAB with a browser (as explained in Section 3.4). The server-side in the case of the scenario is a Raspberry Pi but it can also be a computer or a laptop, it is the machine where openHAB is hosted and is running. As said before, to implement the RBAC model into openHAB two repositories have been modified, the openhab-webui repository and the openhab-core repository. The openhab-webui repository implements all the main parts of the user interface, this is why it appears on the client-side. The openhab-core repository contains the main functionalities of openHAB, therefore it appears on the server-side.

The interaction between the server-side and the client-side is made with the HTTP protocol. Multiple clients can interact with the server at the same time. The client has the possibility to be located inside the network or outside the network. If the client is outside the network, the data are encrypted with a tunnel by using TailScale VPN [79].

4.1.3 Access control management

To authenticate each HTTP request from a user and manage the access control of the platform, openHAB uses Json Web Token (JWT). JWT is an open standard (RFC 7519) that defines a compact and self-contained way to transfer data, encoded as a JSON object, between two parties in a secure manner with a digital signature [4]. The Figure 4.4 below shows the steps for using JWT in openHAB [13, 10].

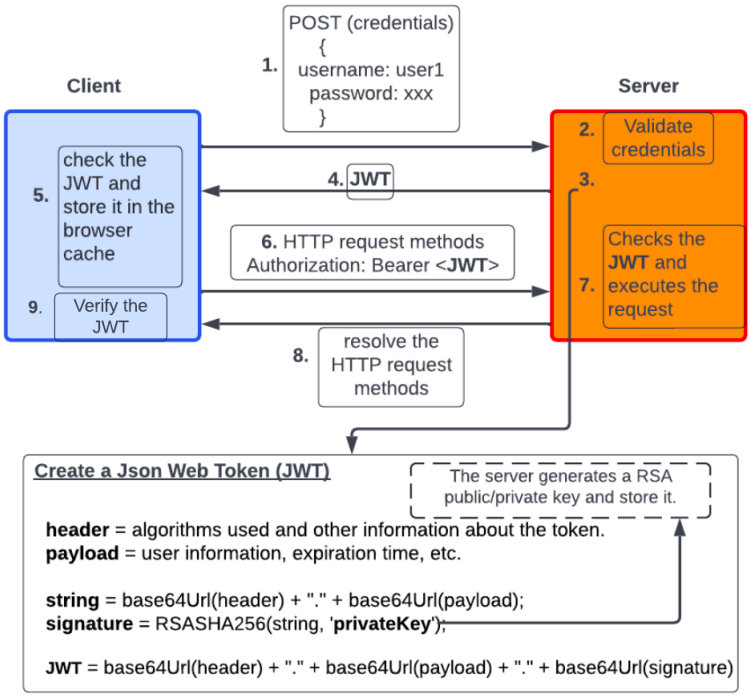

Figure 4.4: steps of JWT into openHAB

The different steps are as follows:

1. The client sends his username and his password to openHAB (server-side).
2. OpenHAB validates the credentials of the user stored on the platform.
3. The representation of a JWT is `header.payload.signature`. OpenHAB creates a JWT with RSASHA256. RSASHA256 is an asymmetric algorithm that creates a signature with the hash function SHA256 [37]. OpenHAB uses the RSA private key to generate the signature. The RSA public key is used to validate the signature. The signature is performed with the Base64Url encoded header, the Base64Url encoded payload, the RSA private key, and the hash function SHA256. The final result of the JWT is the header, the payload, and the signature. The Figure 4.5 shows an example of a JWT where the username is *nico* and its role is the *administrator*.
4. OpenHAB sends the token to the client.
5. The client verifies the token with the RSA public key and stores it in his browser cache.

6. The client for each HTTP requests method such as GET to request resources, HEAD identical to GET but without the response body, POST to submit resources, PUT to replace resources, and DELETE to delete resources puts the JWT received by openHAB previously in the Authorization header of its request [46].

7. OpenHAB verifies the JWT with the RSA public key.

8. OpenHAB resolves the HTTP request method.

9. The client verifies the JWT of the HTTP response with the RSA public key that matches the RSA private key used by openHAB (server-side) to generate the JWT.

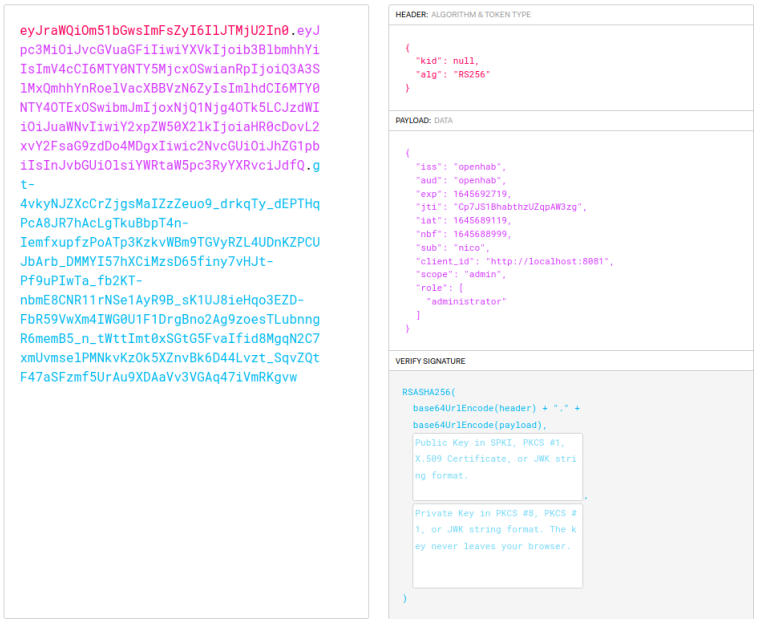

Figure 4.5: Example of a JWT [5]

The signature of the JWT allows openHAB (on the server-side) and the user of openHAB (on the client-side) to verify the integrity of the JWT. The information contained in the JWT payload identifies the user and his authorized access to the platform. Thus, since the integrity is verified, if an attacker tries to modify the payload information, the JWT will not be valid. The signature is created with the RSA private key, so only openHAB will be able to create a signature that matches and can be validated with the RSA public key. The integrity of the JWT header and payload are checked but not encrypted. They are only encoded in Base64Url. Therefore, an attacker can easily decode the payload and the header of the JWT. The main advantage of JWT is that the JWT can easily be embedded in the HTTP request header because its size is small. It is a stateless authentication solution i.e openHAB (server-side) does not need to maintain a session state for each user. Instead, openHAB only sends a JWT for each authenticated user and when a user makes an HTTP request, it puts the received JWT in its request header and openHAB verifies the JWT to authorize and resolve the request.

The architecture from a technical point of view, the client-server model, and the access control management of the openHAB have been described in detail. In the following section, the prerequisites will be listed and explained.

4.2 Prerequisites

The following software and IDE must be installed on the computer in order to modify, build and run the openHAB source code.

- **Git** is used to track changes to the openHAB source code. It is used to coordinate work between programmers. Git is a free, open-source, distributed version control system designed to manage all projects, from the smallest to the largest, quickly and efficiently [81].
- **Maven** is used to build openHAB projects because openHAB bundles have a *Maven* architecture (as explained in Section 4.1.1).
- **Java** is the main language used by openHAB. He uses version 11.
- **Node.js** with version 16.14.0 to be able to run the javascript environment. Node.js uses the non-blocking I/O paradigm, which ensures good performance. Non-blocking I/O allows a single process to serve multiple requests at the same time.
- **npm** with version 8.6.0. Npm allows managing Javascript packages in a Javascript runtime environment.
- Vue.js, webpack and Framework7 to manage the main UI.
- **IntelliJ** is used to have a suitable OSGi/java development.
- **VSCode** is used to easily manage the files of a project.

4.3 Build and run modifications of openHAB

Now the steps to build and run the modification realized in the source code of openHAB in a computer will be explained in four parts. The first part will explain how to run the distribution of openHAB in the localhost[4]. The second part will explain how to debug the openHAB distribution. The third part will describe how to see the changes made to the main interface. The fourth part will explain how to integrate the bundle that has been modified into the instance of openHAB running on the localhost.

OpenHAB documentation [60] recommends three different IDEs to have a suitable OSGi/Java development. There are VSCode, Eclipse or Intellij. For this thesis, the Intellij IDE has been chosen.

[4] *In computer networking, localhost is a hostname that refers to the current device used to access it. It is used to access the network services that are running on the host via the loopback network interface. Using the loopback interface bypasses any local network interface hardware* [84].

A post [53] explained how to build and run openHAB with the modifications realized to the project with the Intellij IDE. Therefore, for the thesis, the steps in the post [53] which explains how to set-up Intellij for openHAB development have been followed.

4.3.1 Run the openHAB distribution

The steps to run the openHAB distribution are as follows:

1. The project openhab-distro has been cloned with the command `git clone` from git
2. The project openhab-distro was built with Maven with the command `mvn clean install`. As shown in the Figure 4.6.
3. The target folders contain the built zip distribution files. For the thesis, the zip distribution file that has been used is the file located at `<DISTRO_DIR>/distribution/openhab/target/openhab-3.3.0-SNAPSHOT.zip`.
4. The zip file openhab-3.3.0-SNAPSHOT.zip has been extracted to the created directory **openhab-test-final-distro**.
5. OpenHAB can be launched with the script `start.sh` or `start_debug.sh` located inside the directory **openhab-test-final-distro**. As you can see in the Figure 4.7 openHAB is running on the localhost. The Figure 4.8 shows that all the bundles are active. With the command `log:tail` it is possible to see the logs of openHAB which are the messages sent in the event bus, as shown in the Figure 4.9 where the simulated temperatures are sent to the event bus.
6. The openHAB console can be reached on the terminal where the script `start.sh` or `start_debug.sh` has been executed and the main UI is available by putting the address `localhost:8080` in a browser of the computer.

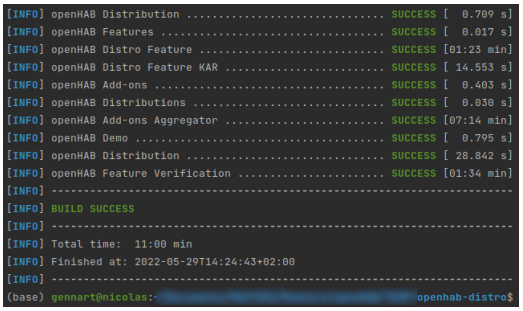

Figure 4.6: building of the openhab-distro project

Figure 4.7: running openHAB

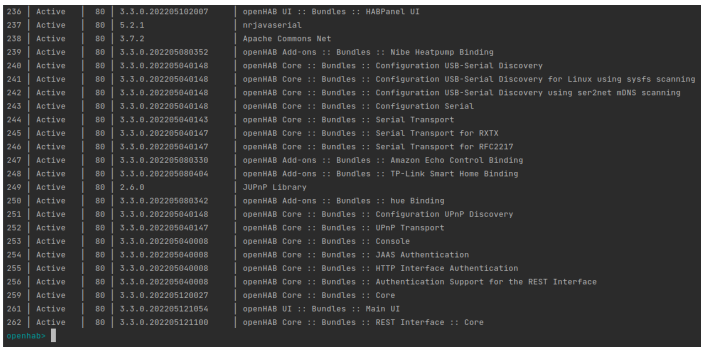

Figure 4.8: List of active bundle

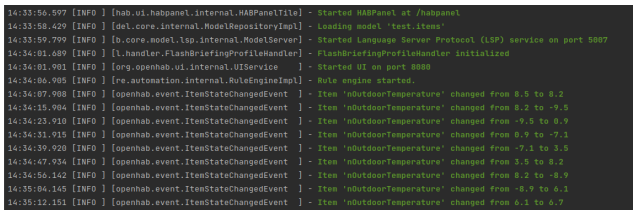

Figure 4.9: openHAB log message

The script **start_debug.sh** allows to debug the platform. For the thesis, the project that has been debugged is the openhab-core project. The way to debug this project is explained just after.

4.3.2 Debug the openHAB distribution

The steps to debug the project openhab-core are the following:

1. The menu *Run* → *Edit Configurations...* has been opened.
2. *+* sign has been clicked to add a *Remote configuration*
3. The host *localhost* and the port *5005* has been used.
4. The *Apply* button has been clicked and then the *OK* button has been clicked.
5. The final configuration is displayed in the Figure 4.10.
6. The Figure 4.11 shows that the openhab-core project is connected to the target virtual machine at the address localhost:5005 to be able to debug the platform.

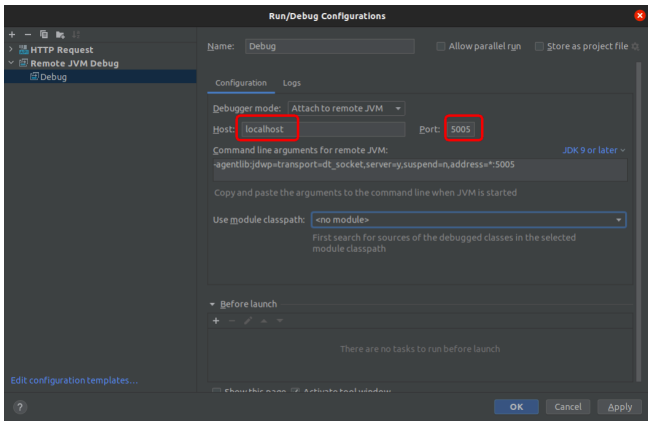

Figure 4.10: Configuration to debug the source code of openHAB

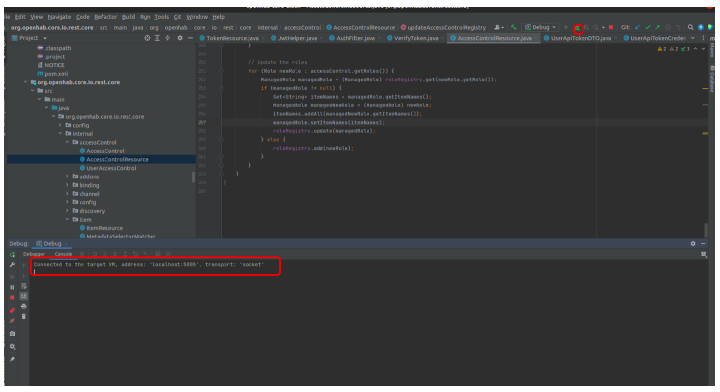

Figure 4.11: Connection of a project to a target VM at the address localhost:5005

4.3.3 See the modifications made to the main UI

The project openhab-webui is used to modify to the main UI. To see the modifications made to the openhab-webui project, an openHAB instance must already be running on the localhost (by launching the script `start.sh` or `start_debug.sh` of the `openhab-test-final-distro` repository as explained above). Once an openHAB instance is running on the localhost the `npm start` command at the `openhab-webui/bundles/org.openhab.ui/web` location is executed. The openhab-webui project manages a lot of file therefore the files monitored by the computer will reached the limit when the `npm start` command is executed. To solve this problem the limit of files that the computer can monitored mus be increased. To increase the number of files monitored by the computer the line `fs.inotify.max_user_watches=524288` at the location `/etc/sysctl.conf` of the Linux operating system must be put. Then the command `sudo sysctl -p` must be executed to load the sysctl.conf file with the new added line.

The Figure 4.12 shows that the user interface is located at the address `localhost:8081` where 8081 is the next available port after the openHAB running instance which uses the port 8080 at the address `localhost:8080`. The Figure 4.13 shows that the project has been successfully compiled with npm and the changes to the user interface are available at `localhost:8081` as displayed at Figure 4.14.

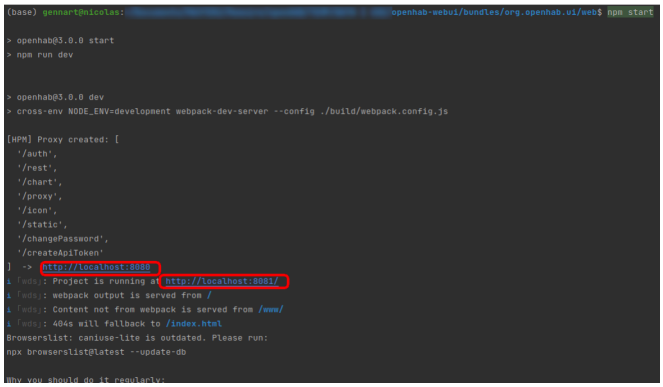

Figure 4.12: The development server available at the port 8081

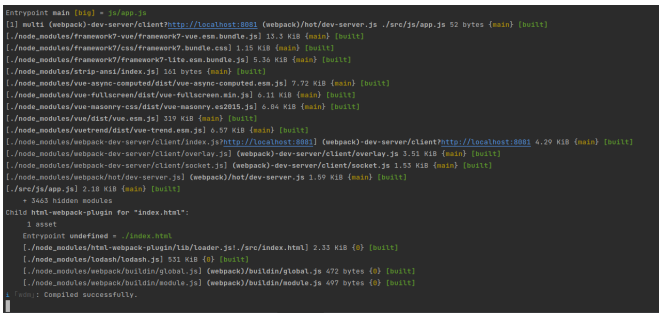

Figure 4.13: Compiled successfully with `npm start`

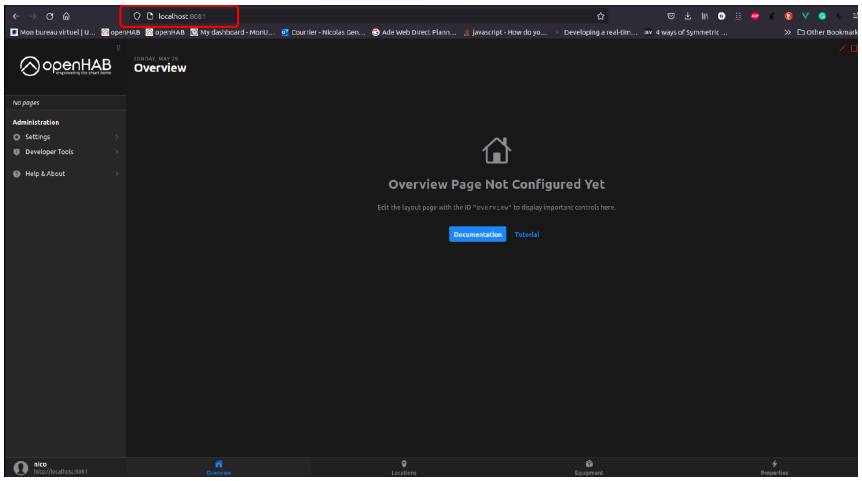

Figure 4.14: Development server reached at the address locahost:8081

4.3.4 Incorporate new bundles into the running instance of openHAB

To implement the RBAC model into openHAB the following bundles (represented as JAR files) from the openhab-core project and the openhab-webui project were modified:

1. **org.openhab.core-3.3.0-SNAPSHOT.jar**
2. **org.openhab.core.auth.jaas-3.3.0-SNAPSHOT.jar**
3. **org.openhab.core.io.console-3.3.0-SNAPSHOT.jar**
4. **org.openhab.core.io.http.auth-3.3.0-SNAPSHOT.jar**
5. **org.openhab.core.io.rest.auth-3.3.0-SNAPSHOT.jar**
6. **org.openhab.core.io.rest.core-3.3.0-SNAPSHOT.jar**
7. **org.openhab.ui-3.3.0-SNAPSHOT.jar**

These bundles contain all the modified code parts of the openhab-core project and the openhab-webui project. The above bundles from point 1 to point 6 are from the openhab-core project and the above bundle from point 7 is from openhab-webui project.

The explanation of the incorporation of one bundle in the running instance of openHAB will be done. Then the same steps will be performed for the other bundles. The detailed explanation will be realized for the bundle **org.openhab.core** from the openhab-core project.

First, the JAR file that represents the bundle **org.openhab.core** must be generated. A JAR file is the aggregation of java classes and associated metadata and resources, (see Section 4.1.1). Building a bundle is done with *Maven* as this is the infrastructure used by openHAB to manage bundles as described above in Section 4.1.1. The `mvn clean install` command must be run at the `bundles` location of the openhab-core project.

The Figure 4.15 shows that the project has been successfully built. A JAR file will be generated in each target directory of each bundle of the project openhab-core as shown in the Figure 4.16. Each time, the whole project must be built. If the whole project is not built, the JAR files will not be created correctly because of the dependencies between the bundles. Therefore, it will take a lot of time to build the whole project for each code change each time. However, it is possible to optimize the project build by running the following command:

```
mvn clean install -DskipChecks=true -DskipTests -Dfeatures.verify.skip=true
-T 1C
```

`<-DskipChecks=true>` to skip the checks, `<DskipTests>` to skip the tests, `<-Dfeatures.verify.skip=true>` to skip the features verification and `-T 1C` to specify the thread count where C is core multiple.

Often, the `mvn spotless:apply` command must be executed to apply the required format for the openHAB Java files as shown in the Figure 4.17. There is an additional step to do for the bundle **org.openhab.ui** which comes from the openhab-webui project. The step is to run the command `npm run lint:fix` at the location `openhab-webui/bundles/org.openhab.ui/web` to fix the required format of openHAB for the Vue.js files.

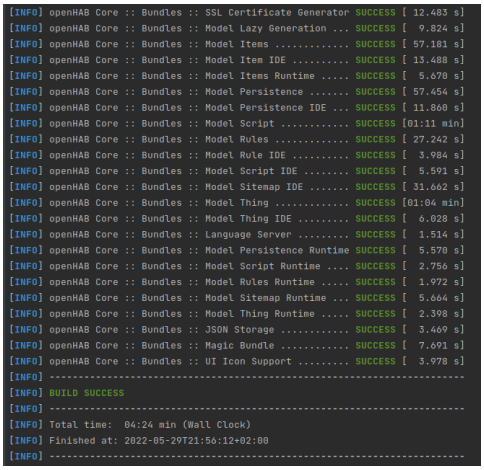

Figure 4.15: Building the openhab-core successfully

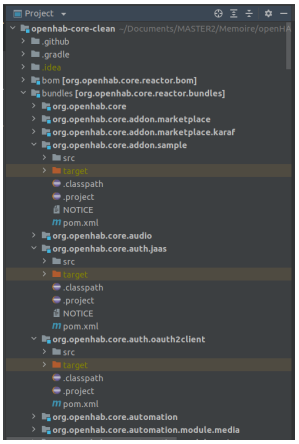

Figure 4.16: Generation of `target` directories in each bundles

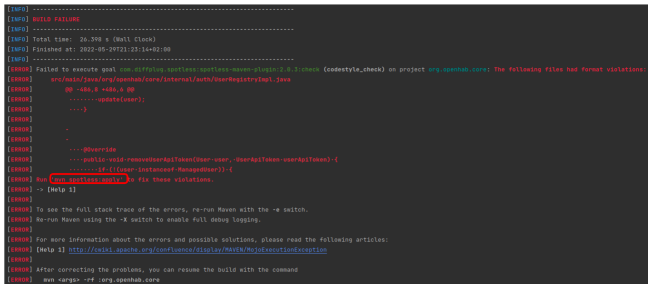

Figure 4.17: The mandatory format is required for the openhab-core project

To properly place the bundle **org.openhab.core** in the running instance of openHAB, the following steps must be performed:

1. The old bundle has been uninstalled as shown in the Figure 4.18.

2. The jar file of the modified bundle at the location `openhab-test-final-distro/addon` has been placed as shown in Figure 4.19.

3. After a few seconds the modified bundle will be in the state ACTIVE with another bundle ID which is 263 as shown in Figure 4.20.

4. If the bundle remains in the RESOLVED or WAITING state, the running instance of openHAB must be stopped and restarted by running the `./start.sh` or `./start_debug.sh` script in the `openhab-test-final-distro` directory. By restarting the running instance of openHAB the modified bundle will be in the ACTIVE state. The Figure 4.21 shows that the bundle **org.openhab.core** has the state ACTIVE. Therefore the bundle **org.openhab.core** has been well replaced in the running instance of openHAB.

The above steps are the same for other bundles. These steps must be done bundle by bundle and not by adding all bundles at once, otherwise it will cause problems with the runtime environment.

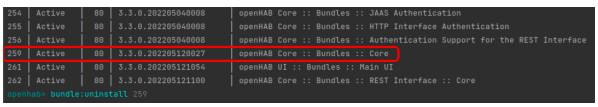

Figure 4.18: Uninstallation of the bundle **org.openhab.core**

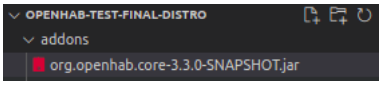

Figure 4.19: JAR file of the bundle **org.openhab.core** at the location `openhab-test-final-distro/addon`

Figure 4.20: The bundle **org.openhab.core** with another ID

Figure 4.21: The bundle **org.openhab.core** with the state ACTIVE

4.4 Persistent roles and groups management

The steps to build and execute changes to openHAB have been explained. This section will present how roles and groups are managed in openHAB in a persistent manner.

The Figure 4.22 shows a Java class diagram for managing persistence roles and groups. The diagram is only composed of classes that manage users, groups and roles and not other classes that manage Items for example. The Java class diagram contains two bundles, the bundle **org.openhab.core** and the bundle **org.openhab.core.storage.json**. The **org.openhab.core** bundle is composed of the set of classes (in the orange frame) that define the User, Group, and Role objects, the set of classes (in the blue frame) that represent the registries of users, groups, and roles, the set of classes (in the green frame) that interact with the files that store users, groups, and roles in JSON format, and many other classes that will not be listed. The bundle **org.openhab.core.storage.json** consists of the set of classes (in the red frame) that store/obtain data in JSON format to/from a file and other classes that are not shown.

The orange frame in the Figure 4.22 contains the classes that define the User, Group and Role objects. Previously, there were no objects that defined a role or a group. Therefore, the classes `ManagedGroup` and `ManagedRole` and the interfaces `Group` and `Role` were added. The `ManagedUser` class has been modified to also manage groups for a user. A User object contains a set of groups and a set of roles to specify the roles and groups that a user contains. A Group object contains a set of roles to specify the roles that a group contains. A Role object contains a set of Item names. Each Item has a unique name (see Section 3.3.1) and an Item (see Section 3.1) represents a specific device, location, equipment, or property and the information it contains. Therefore, the set of Item names within the Role object specifies the authorized resources (Items) that the role will assign to users who owns the role.

The classes in the blue frame represents the registries to manage users, groups and roles in openHAB that can be accessed from other bundles and classes within the platform. The classes `RoleRegistryImpl`, `GroupRegistryImpl`, and `UserRegistryImpl` implement functions for getting, adding, deleting, and updating roles, groups, and users respectively in openHAB in a persistent manner. This is done persistently because the registries use the classes in the green frame of the Figure 4.22 that interact with the files that store the users, groups and roles in JSON format. The interaction is done by using the `ServiceStorage` service of the **org.openhab.core.storage.json** bundle that handles JSON-formatted data stored in persistent files inside openHAB. In openHAB only users were managed persistently, so to manage groups and roles per-

sistently as well, the classes `RoleRegistryImpl`, `GroupRegistryImpl`, `RoleProvider`, `GroupProvider`, `ManagedRoleProvider` and `ManagedGroupProvider` and the interfaces `Roleregistry` and `GroupRegistry` were added and implemented.

4.5 Verify JWT and filter resources

As noted in the Section 2.5.1, all resources are returned for each request from each openHAB user. There was no access control. Therefore, this section explains how resources are filtered for each request from each openHAB user with roles and groups and how openHAB access control is handled with the RBAC model.

The steps to return the resources allowed to a user for each of his requests are as follows:

1. As explained in the Section 4.1.3, a JWT is placed in the header of each HTTP request method. Therefore, the JWT placed in the Authorization header of the user's request will be checked and the username contained in the JWT will be retrieved.

2. On the server side (see Section 4.3), once the JWT has been validated and the username contained in the JWT payload has been extracted, the user's roles and groups can be obtained with the user registry (see the `UserRegistry` interface in the Figure 4.22).

3. If the user has groups, the roles that belong to his groups will be obtained with the group registry (see the interface `GroupRegistry` of the Figure 4.22).

4. With the combination of the roles obtained from the groups and the user's roles, the set of authorized Items names will be collected with the role registry (see the interface `RoleRegistry` of the Figure 4.22).

5. The set of Items names obtained will be returned to the user who made the request.

6. The user (on the client side, see Section 4.3) will only see in his interface the resources that correspond to his authorized Items.

For requests made by a user where the username extracted in the JWT payload does not exist, the JWT is invalid, the user has no role and no group, or the user has only the role *user* no resources (Items) will be returned to him. However, if the *administrator* role is in the JWT payload of a request and the JWT is valid, all resources (Items) will be returned to the request.

For this part, three bundles have been modified. The bundle **org.openhab.cor.io.rest.auth** to have a function which verifies a JWT and extracts the payload information from the JWT (realized in the class `VerifyToken`). The **org.openhab.core** bundle implement the function (in the `ItemRegistry` class) that returns the authorized Items for the user. The bundle **org.openhab.core.io.rest.core** to return the authorized Items for each request made by a user based on their roles and groups (these changes were implemented in the

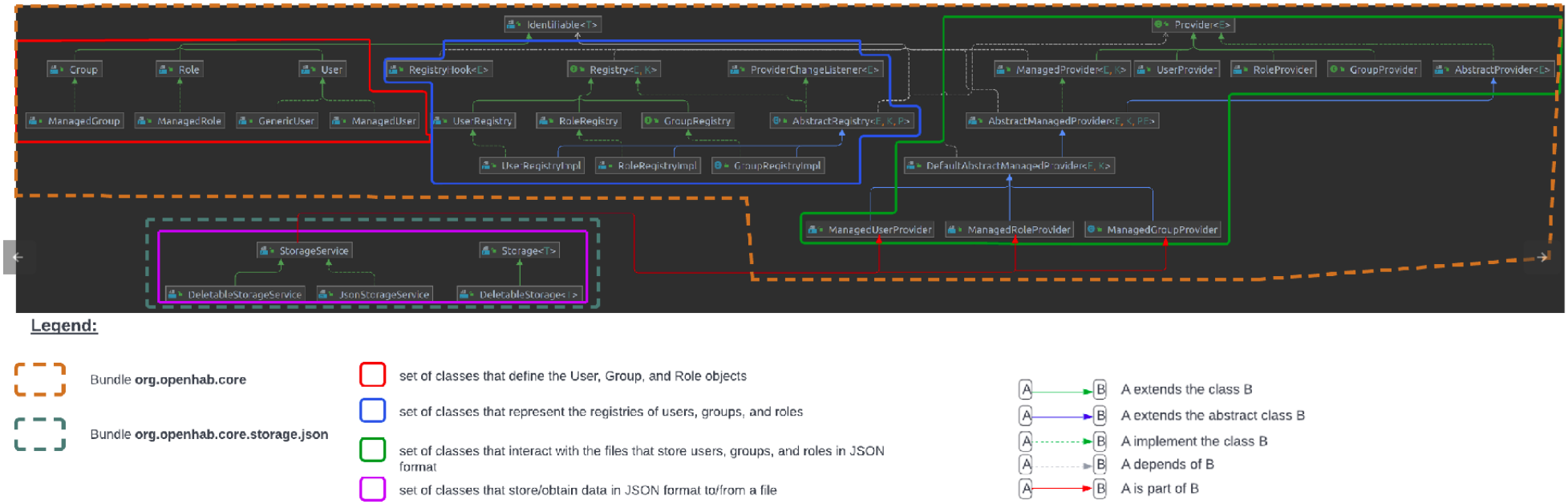

Figure 4.22: Classes diagram for persistent roles and groups management

`ItemResource` class).

Each user of openHAB has access to the platform's resources based on their roles and groups where each role and group specifies the authorized resources assigned to users who own the role or group. As a result, the conclusion is that the RBAC model has been incorporated into openHAB.

4.6 Role-based access control management

The RBAC model has been incorporated into openHAB, users, groups and roles are stored and managed persistently and resources sent to a user are filtered with his roles and groups. But how to define a new group or a new role? How is it possible to add a role or a group to a user? How is it possible to add a role to a group or an item name to a role? This Sectionpresents how these features have been implemented in the openHAB console and in the main UI.

4.6.1 OpenHAB console

The openHAB instance with the embedded RBAC model runs on the localhost. Thus, the openHAB console can be reached on the terminal where the `start.sh` or `start_debug.sh` script was executed (see Section 4.3.1). With the openHAB console, it is possible to manage the RBAC for openHAB users completely (as described in Section 2.8). The data management is done in a persistent way, which means that even if the running instance of openHAB is stopped, the data will remain in openHAB (the data is stored in a file). The Figures 4.23 and 4.24 show the `openhab-users` command and the `openhab-ac` command with the possible arguments that each of the commands supports.

The functions that has been added to the `openhab:users` command are as follows:

- List the available roles for a user.
- Change the role of a user.
- Add a role to a user.
- Delete a role of a user.
- List the available groups for a user.
- Change the group of a user.
- Add a group to a user.
- Delete a group of a user.

For each of the above functions, when information is modified for a user, it is updated in the user registry (with the `UserRegistry` interface, see Section 4.4). In the user registry, there must always be at least one user with the role `administrator`. It is therefore not possible to change the role *administrator* with another role or to delete the role

administrator if there is only one user with the role *administrator* in the user registry. This prevents the deletion of all users with the role *administrator*.

Before the command `openhab:ac` did not exist in openHAB. The command was added with the following functions:

- Display the roles assigned to each group and the Item names assigned to each role.
- List the available groups.
- Change a group.
- Add a group.
- Remove a group.
- Add a role to a group.
- remove a role to a group.
- List the available roles.
- Change a role.
- Add a role.
- Remove a role.
- Add an Item name to a role. When an item name is added to a role, all children of this Item name are also added, in order to respect the semantic model (see Section 3.5). For example, in the scenario (see Section 2.5.2), if the Item name *gOutdoor* is added to the role, the Item name *nOutdoorTemperature* that represents the outdoor temperature is also added to the role.
- Remove an Item name to a role. When an item name is removed from a role, all children of this Item name are also removed, in order to respect the semantic model (see Section 3.5). For example, in the scenario (see Section 2.5.2), if the Item name *gOutdoor* is removed from the role, the Item name *nOutdoorTemperature* that represents the outdoor temperature is also removed from the role.

For each of the above functions, when a group's information is modified, the group is updated in the group registry (with the interface `GroupRegistry`, see the Section 4.4), when a group is added, deleted or modified, the modifications are also made in the group registry. For functions that manage roles, changes are also made in a registry, except that it is in the role registry (with the `RoleRegistry` interface, see the Section 4.4).

Previously, every user could perform the functions available from the `openhab:users`. This is unsafe because it means that every user, regardless of role, can delete or add a user (even users with the role *administrator*) and add a role or group to a user to allow access to resources. Therefore, the password and username of a user with the role *administrator* must be entered in order to perform any of the functions in the `openhab:users` command

and in the created command `openhab:ac` (except for the function that makes no changes in the registry such as the list of available users, groups or roles). This check ensures that only users who know the password of a user with the *administrator* role can manage users and user access control for the platform. The Figure 4.25 shows that if a user wants to add the role *role_child1* to the user *child1* the password of a user with the role *administrator* must be set.

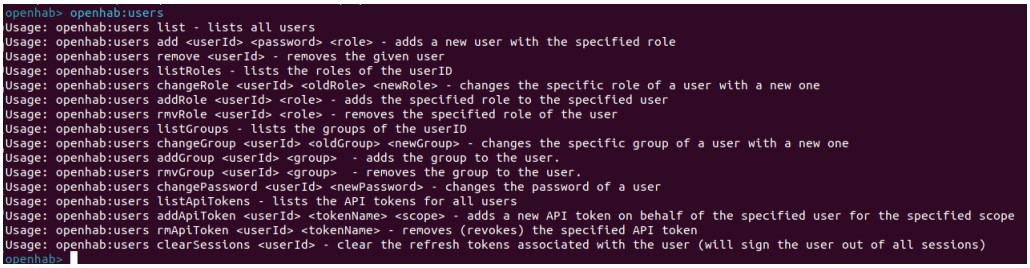

Figure 4.23: `openhab-users` command

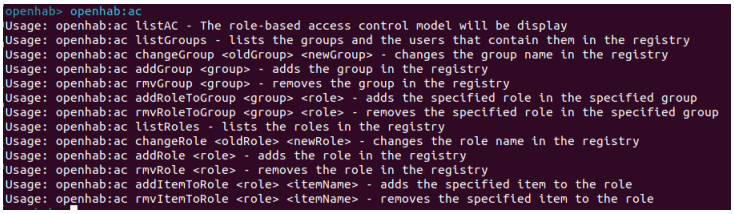

Figure 4.24: `openhab-ac` command

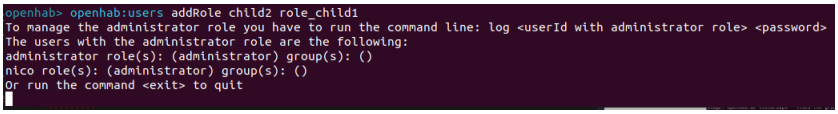

Figure 4.25: Verification of the password of a user with the role of *administrator*

For this part, the bundle **org.openhab.core.io.console** with the class `ACConsoleCommandExtension` and the class `UserConsoleCommandExtensionhas`[5] has been modified.

4.6.2 Main UI

As mentioned in the previous section, the instance of openHAB with the modifications is hosted and runs on the localhost. Thus, the main UI is available by putting the address `localhost:8080` in a browser on the computer (as explained in Section 4.3.1). For time reasons, in the main UI, it is only possible to create a role and a group. However, all users with their roles and groups, all groups with their roles, and all roles with their Item names

[5]The classes `ACConsoleCommandExtension` and `UserConsoleCommandExtensionhas` are at the location `openhab-core/bundles/org.openhab.core.io.console /src/main/java/org/openhab/core/io/console/internal/extension`

are displayed.

The Figure 4.26 shows the information that manages the access control of openHAB based on the RBAC model. It is possible to create a role or a group (see Figure 4.27). Afterward, by clicking on the *Save* button (see Figure 4.26), all the roles and groups that have been created with the main interface will be stored in the server (see Section 4.3). The Figure 4.29 shows the available users with their roles and groups. The Figure 4.28 shows the available groups with their roles. The Figure 4.30 shows the available roles with their Item names. As you can understand, the Figures below display the changes to the main openHAB interface. They show in the main UI the access control of the scenario users (see Section 2.5.2) based on the RBAC model.

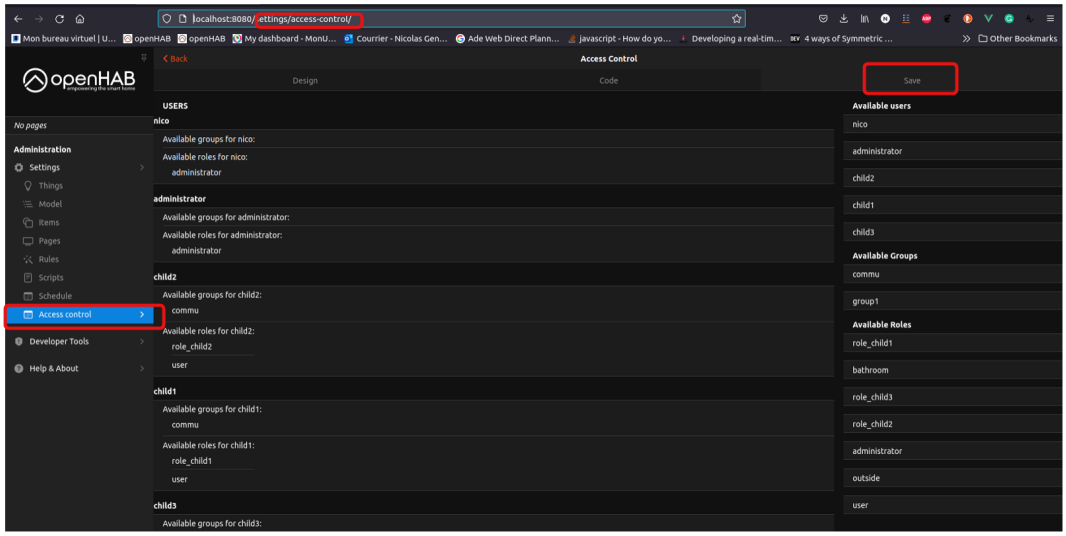

Figure 4.26: Access control page in the main UI

Figure 4.27: Creation of a role or a group with the main UI

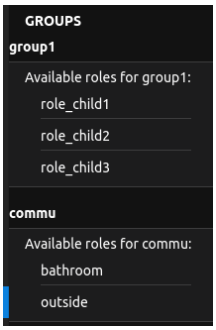

Figure 4.28: Available groups with their roles

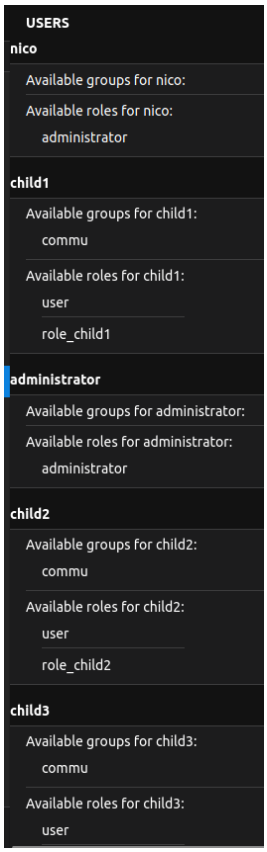

Figure 4.29: Available users with their roles and groups

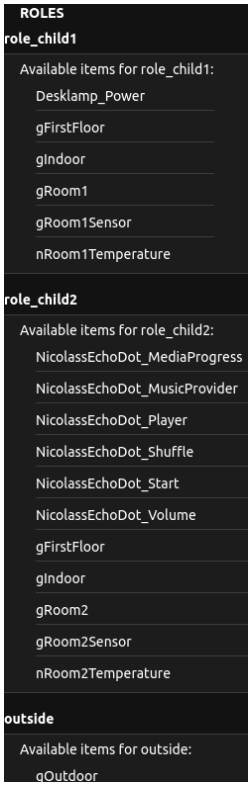

Figure 4.30: available roles with their Item names

The management of the main UI on the client side (see Section 4.3) is done with Vue.js, webpack and Framework7 (see Section 4.2). The bundle **org.openhab.ui** from the project openhab-webui has been modified . The *Vue.js* file `roles-management.vue`[6] has been created and implements the display of the access control of the platform based on the RBAC model (as shown in the Figures 4.27, 4.28, 4.29, 4.30). The *Javascript* file `routes.js`[7] has been modified to create the route `/settings/access-control/` (see Figure 4.26) in the client side to access the page that displays the access control of the platform based on the RBAC model. Finally the *Vue.js* file `app.vue`[8] has been modified to put the *access-control* button in the settings menu of the main UI (see Figure 4.26). On the client side, the functions that implement the HTTP methods (see Section 4.3) were already implemented, so nothing had to be changed for this part.

On the server-side new REST resources[9] has been added to enable the client-side to

[6]The *Vue.js* file `roles-management.vue` is at the location
`openhab-webui/bundles/org.openhab.ui/web/src/pages/settings/access-control/roles-management.vue`

[7]The *Javascript* file `routes.js` is at the location
`openhab-webui/bundles/org.openhab.ui/web/src/js/routes.js`

[8]The *Vue.js* file `app.vue` is at the location
`openhab-webui/bundles/org.openhab.ui/web/src/components/app.vue`

[9]*A REST resource can be defined as a vital element to be referenced within the client-server system.*

get/put users with their roles and groups, groups with their roles, and roles with their Item names. The REST resources are the `rest/accessControl` and the `rest/accessControl/put`. The REST resource `rest/accessControl` accepts the HTTP method GET such that the client-side can get RBAC information. This HTTP GET request is made when the user (on the client-side) goes to the address `<openHAB address>/settings/access-control/` (see Figure 4.26) or when the user is already at this address and refreshes the page. The REST resource `rest/accessControl/put` accepts the HTTP method PUT such that the client-side can put RBAC information on the server-side. This HTTP PUT request is made when the client-side creates a role or a group and saves it by clicking on the *Save* button (see Figure 4.26). An example of data put in the body of the response of the HTTP GET request and put in the body of the HTTP PUT request is shown below:

```
{accessControl: {
    userAccessControlSet: [{
        name: 'nico',
        roles: ['role1', 'role2'],
        groups: ['group1']
      }],
    groups: [{ group: 'group1', roles: ['role1', 'role2'] },
             { group: 'group3', roles: ['role1', 'role2'] }],

    roles: [{ role: 'role1', itemNames: ['item1', 'item2'] },
            { role: 'role2', itemNames: ['item3', 'item4'] },
            { role: 'role3', itemNames: ['item3', 'item2'] }]
    }
}
```

The data contains the username of each user and their roles and groups, the available groups with their roles, and the available roles with their Item names. When this data is either put on the server or got from the server, the JWT in the HTTP header of the request/response is checked and the presence of the role *administrator* in the JWT payload is verified (see Section 4.5).

For security reasons only users with the role of *administrator* can access the page in the main UI (see Figure 4.26) that displays the access control of openHAB.

For the server side the bundle **org.openhab.core.io.rest.core** has been modified where the classes `AccessControlResource`, `AccessControl` and `UserAccessControl`[10] have been added and implemented.

REST architecture treats all of its content as a resource, which includes Html Pages, Images, Text Files, Videos, etc. Access to resources is provided by the REST server where REST client is used for accessing as well as modification of resources. All of its resources get identified via URI, which is abbreviated as Uniform Resource Identifier [87].

[10]The classes `AccessControlResource` `AccessControl` and `UserAccessControl` are at the location `openhab-core-clean/bundles/org.openhab.core.io.rest.core /src/main/java/org/openhab/core/io/rest/core/internal/accessControl`

4.6.3 Remaining improvements to the main UI

The Figure 4.31 shows how the page that handles the RBAC model in the main UI should look to properly handle RBAC for openHAB users with the UI. Due to time constraints, it was not possible to complete the full implementation of RBAC management with the main UI. Figure 4.31 represents the Figure 4.26. What are the missing features for the main UI regarding Figure 4.31? In this section, the missing features will be listed and explained. Then, the beginning of functionality will be described which is an interpreter to check a specific syntax that allows defining RBAC for openHAB users with sentences.

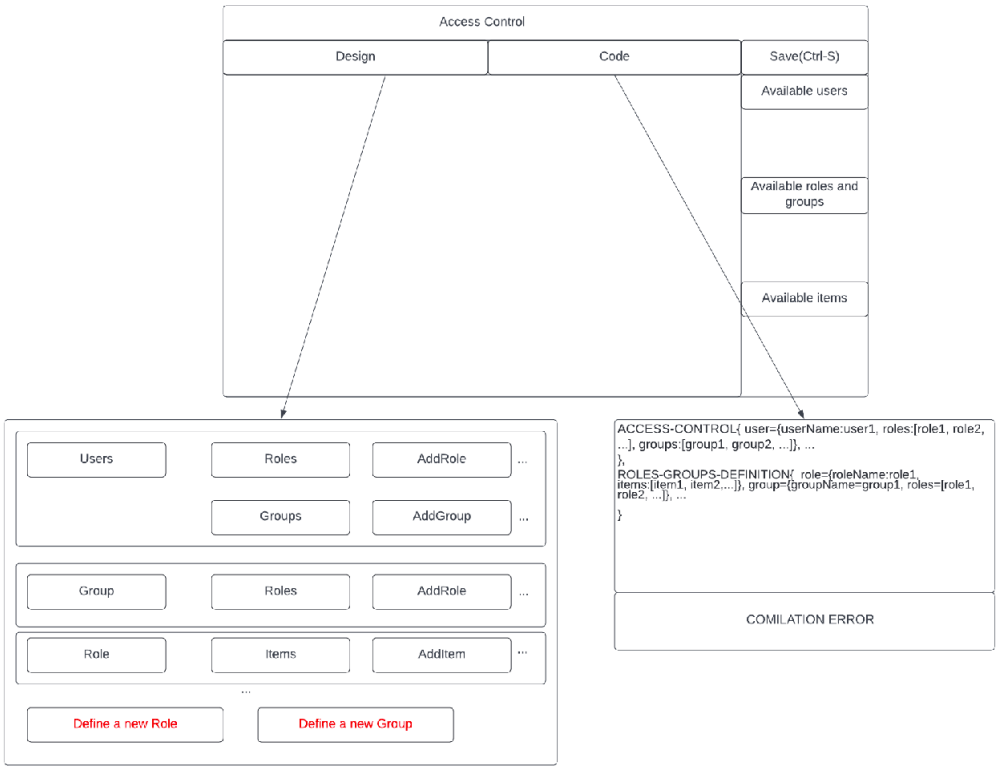

Figure 4.31: the objective for the main UI

The missing features for the **Design** part of the Figure 4.31 are as follows:

- Add a role to a user.
- Remove a role of a user.
- Add a group to a user.
- Remove a group of a user.
- Add a role to a group.
- Remove a role to a group.
- Add an Item name to a role.

- Remove an Item name to a role.

Removing a user and deleting a user should also be implemented in the main UI but in a different place.

The purpose of the **Code** part is to be able to define the RBAC model with sentences that respect a specific syntax. The **Code** is divided into two parts (as shown in the Figure 4.31). The first part starts with the word ACCESS-CONTROL. This first part is for assigning existing roles and groups to exist users. The second part starts with the word ROLES-GROUPS-DEFINITION. This second part is for defining new groups and roles by assigning roles and Item names to them respectively. When a user accesses the **Code** location of the main UI, the RBAC for openHAB users that is already present in openHAB should be automatically generated in a sentence that follows the defined syntax (as shown in Figure 4.31). An interpreter must be implemented to verify that the syntax for defining RBAC for openHAB users is valid and that there is no confusion in defining elements, such as defining a role and a group with the same string or assigning a role that has not been defined to a user.

The beginning of an interpreter has been realized. The project uses two packages which are nearley and moo which can be installed with **npm** (see Section 4.2). Nearley consists of a compiler and a parser. The nearley compiler converts the grammar definition of a simple BNF-based syntax into a small JS module. The nearley parser uses the created JS module to parse a string. Moo consists of a highly optimized tokenizer/lexer generator that is used in the defined grammar to tokenize strings [49, 48].

The grammar that was written is in the file `json.ne` with a BNF-based syntax of the created project. To compile the nearley grammar, the command `nearleyc json.ne -o json.js` must be executed. Then, the `parser.js` file (in the project) uses the created `json.js` file and a nearley parser to parse a string. The `parser.js` file is executed with Node.js (see Section 4.2) with the `node parser.js` command.

For example, the string below:

```
group=(groupName=group1 ,
    roles={role=(roleName=role1 ,accessTo=[item1 ,item2 ]) ,
role=(roleName=role2 ,accessTo=[item2 ,item3 ])}) ,
role=(roleName=role3 ,accessTo=[item1 ,itemt2 ])
```

aims to define a group with the name *group1* that contains a role with the name *role1* with some Item names, a role with the name *role2* with some Item names, and a role with the name *role2* with some Item names. The result of the parser is displayed in the Figure 4.32.

If the syntax is not respected, by the absence of parenthesis, for example, the parser returns an error as shown in the Figure 4.33.

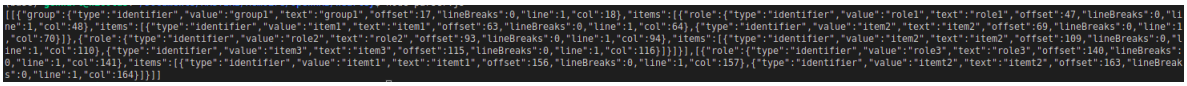
[[{"group":{"type":"identifier","value":"group1","text":"group1","offset":17,"lineBreaks":0,"line":1,"col":18},"items":[{"role":{"type":"identifier","value":"role1","text":"role1","offset":47,"lineBreaks":0,"line":1,"col":48},"items":[{"type":"identifier","value":"item1","text":"item1","offset":63,"lineBreaks":0,"line":1,"col":64},{"type":"identifier","value":"item2","text":"item2","offset":69,"lineBreaks":0,"line":1,"col":70}]},{"role":{"type":"identifier","value":"role2","text":"role2","offset":93,"lineBreaks":0,"line":1,"col":94},"items":[{"type":"identifier","value":"item2","text":"item2","offset":109,"lineBreaks":0,"line":1,"col":110},{"type":"identifier","value":"item3","text":"item3","offset":115,"lineBreaks":0,"line":1,"col":116}]}]},{"role":{"type":"identifier","value":"role3","text":"role3","offset":140,"lineBreaks":0,"line":1,"col":141},"items":[{"type":"identifier","value":"itemt1","text":"itemt1","offset":156,"lineBreaks":0,"line":1,"col":157},{"type":"identifier","value":"itemt2","text":"itemt2","offset":163,"lineBreaks":0,"line":1,"col":164}]}]]

Figure 4.32: result of the parser

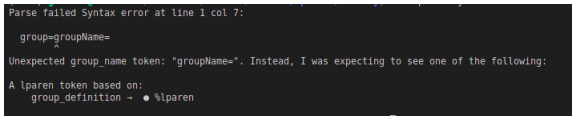

Figure 4.33: Parser error

To conclude, the remaining goals for the main UI are not essential because RBAC for openHAB users can be fully managed with the openHAB console and there are already some features implemented in the main UI like displaying roles, groups, etc. However, by implementing the missing functionality for the main UI, RBAC management for openHAB users will be more user-friendly and understandable.

4.7 Use of RBAC for the scenario

The scenario described in Section 2.5.2 is a house with a father and three children where the father has the role of *administrator* and the children have the role of *user* (see Section 2.5.2). The scenario has been configured for the instance of openHAB that contains the changes. This instance of openHAB running on the localhost of a computer has the scenario configured by following the steps in the Section 3.3.4 which explains how to configure the scenario in openHAB. OpenHAB currently has no access control (see Section 2.5.1). Therefore, as illustrated with the scenario in Figure 2.4 that will cause some problems among children. There are quarrels between the children because each child can access all the devices in the smart home (see Figure 2.4). Therefore, one child may disturb his brother by turning on the lights at night. However, in the previous sections, explanations were made on how the RBAC model was integrated into openHAB and how RBAC management for openHAB users with the openHAB console and with the main UI was added to openHAB. Therefore, this section will present how RBAC for openHAB users can solve the problems encountered in our scenario.

The RBAC for openHAB users has been configured for the instance of openHAB that runs on the localhost. The Figure 5.8 shows how the RBAC model has been configured. According to the scenario (see Section 2.5.2) below there are the users with their roles and groups:

- **administrator** → role(s): (administrator) group(s): ()
- **child1** → role(s): (role_child1, user) group(s): (commu)
- **child2** → role(s): (role_child2, user) group(s): (commu)
- **child3** → role(s): (role_child3, user) group(s): (commu)

There are the following groups with the roles they have:

- **group1**: (role_child1, role_child2, role_child3)
- **commu**: (bathroom, outside)

and there are the following roles with the Item names they have below. An Item name identifies an Item (see Section 4.5), and the functionality of each Item below is explained in the Section 3.3.4.

- **administrator** : (has access to all items)
- **user** : (has no access)
- **outside**: (gOutdoor, nOutdoorTemperature)
- **bathroom**: (gBathroom, gBathroomSensor, gOutdoor, nBathroomTemperature, nOutdoorTemperature)
- **role_child1**: (Desklamp_Power, gFirstFloor, gIndoor, gRoom1, gRoom1Sensor, nRoom1Temperature)
- **role_child2**: (NicolassEchoDot_MediaProgress, NicolassEchoDot_MusicProvider, NicolassEchoDot_Player, NicolassEchoDot_Shuffle, NicolassEchoDot_Start, NicolassEchoDot_Volume, gFirstFloor, gIndoor, gRoom2, gRoom2Sensor, nRoom2Temperature)
- **role_child3**: (Huecolorlamp_Color, Huecolorlamp_ColorTemperature, Huewhitelamp1_Brightness, Huewhitelamp2_Brightness, gFirstFloor, gIndoor, gRoom3, gRoom3Sensor, nRoom3Temperature)

With the above configuration, regarding the Figure 2.3 that illustrates the scenario, the conclusion is that the user *administrator* has access to all the devices of the smart home (see Figure 5.2), the user *child1* has access to the devices of the room1, the bathroom, and the outdoor (see Figure 5.3), the user *child2* has access to the devices of the room2, the bathroom and the outdoor (see Figure 5.4), the user *child3* has access to the devices of the room3, the bathroom and the outdoor (see Figure 5.5). There will be no more quarrels between children, as each child will have access to his or her own devices.

In addition, the RBAC model incorporated in openHAB is fine-grained access control because access to a single feature of a device for a user can be removed or added. In fact, with the scenario, if the father wants to prevent the *child2* from starting music with the Amazon Echo device platform at night, he can remove the Items with the following name: **NicolassEchoDot_Player**, **NicolassEchoDot_MusicProvider**, and **NicolassEchoDot_Start** that start music for the Amazon Echo device. These Items are removed for the role **role_child2** with the `openhab:users rmvItemToRole role_child2 <ItemName>` command from the openHAB console (see Section 4.6.1). As a result, user *child2* will not be able to play music from the Amazon Echo device with the platform. The Figure 5.6 shows that the user *child2* can start the music from the Amazon Echo device and then it will not be able as shown in the Figure 5.7.

The openHAB instance with the embedded RBAC model runs on the localhost of a computer. However, it is also possible to run the openHAB instance with the modifications and the configured scenario in a Raspberry Pi. By hosting it on the Raspberry it will be possible to reach it everywhere with Tailscale VPN [79] and at any time because the Raspberry Pi never stops running except if it has been unplugged. Indeed, by following the steps above, it is possible to run the openHAB instance with the RBAC model incorporated and the scenario.

1. The connection to the Raspberry Pi via SSH has been established (see Section 3.4)
2. The openHAB distribution has been cloned and built on the Raspberry Pi. (see Sections 4.3.1 and 4.3.4)
3. The openHABian process on the Raspberry Pi was killed with the command `sudo kill <processIdOfOpenHABian>` executed on the remote terminal.
4. The openHAB distribution built on the Raspberry Pi has been launched (see Section 4.3.1)
5. The bundles that have been modified for the incorporation of the RBAC model in openHAB have been put in the instance of openHAB running on the Raspberry (see Section 4.3.4)
6. The scenario has been configured in the instance of openHAB running on the Raspberry Pi(see Section 3.3.4)
7. The RBAC for openHAB users has been configured according to the scenario (see Section 2.5.2) in the instance of openHAB running on the Raspberry as explained just above (see Sections 4.6.1 and 4.6.2).

In conclusion, RBAC for openHAB users is very useful if the smart home is composed of multiple users and configured with openHAB because access control to resources for each user of the smart home can be managed.

4.8 Challenges

As mentioned in the introduction (see Section 2.8) incorporating the RBAC model into openHAB was not an easy task. Many problems have been encountered. Therefore, these problems will be explained here.

4.8.1 Adding an external package/library into a bundle

Adding an external package inside a bundle will not be performed with the IDE by downloading the external package for the bundle. Instead, a dependency must be added to the bundle that requires the external package. The new dependency into the bundle must contain the required maven repository that specifies the external package (maven repository because the bundles of the platform have a *Maven* architecture as said in the Section 4.1). For example the external package `com.google.gson` has been added for the bundle

org.openhab.core.io.rest.core to be able to use functions that manage JSON object inside the bundle. The external package has been added to the bundle by adding a dependency to it. The dependency specifies the maven repository of the external package [74]. The dependency added to the `pom.xml` file of the bundle **org.openhab.core.io.rest.core** is shown below:

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.9</version>
</dependency>
```

4.8.2 Framework error

A question has been posted on the openHAB community [64] to know why during the execution of the openHAB distribution (see Section 4.3) there was an error with the framework and the runtime environment. Because of the framework error, nothing was working for the running instance of openHAB that contain the modifications. The error occurred when the openHAB project has been updated. To fix the error a thread on the openHAB community [68] and an issue created for the openhab-distro project [29] were used and followed. The framework error has been solved by downloading the KAR[11] file [73] and by putting the downloaded KAR file at the location `openhab-test-final-distro/runtime/system/org/apache/karaf/deployer/org.apache.karaf.deployer.kar/4.1.3`. The directory `openhab-test-final` is the directory of the openHAB distribution (see Section 4.3). The problem with Apache in the openHAB distribution has been successfully corrected.

4.8.3 Internal package

As explained in the Section 4.5, the `VerifyToken` class that verifies the JWT has been added to the **org.openhab.core.io.rest.auth** bundle. Then, the created class should be used in the bundle **org.openhab.core.io.rest.core** to check the JWT for each HTTP request made by a user (client side) to openHAB (server side). The `VerifyToken` class has been placed in the `internal` package of the **org.openhab.core.io.rest.auth** bundle. However, if the class is intended for external use, it must be placed outside the bundle's `internal` package. For this problem, a post was created on the openHAB community to understand why the `VerifyToken` class was not available for other bundles. The reason is that the `VerifyToken` class was placed in the `internal` package of the **org.openhab.core.io.rest.auth** package. By placing the `VerifyToken` class outside of the `internal` package of the **org.openhab.core.io.rest.auth** bundle, the `VerifyToken` class that acts as a service will be available to other openHAB bundles.

[11] *A KAR file is essentially a jar (so a zip file) which contains a set of feature descriptor and bundle jar files [3]*

4.8.4 Service deadlock

A service can depend on another service located in the same bundle. For example, if service A depends on service B, service A can not be ready as long as service B is not ready. A loop has been created between two services. Indeed, for the bundle **org.openhab.core** when the role registry[12] has been created (see Section 4.4) it requires the service Item registry[13] to get the Items. However, the item registry also needs the service role registry to implement the function that filters items with roles (see Section 4.5). Thus, there is a service deadlock between the role registry service and the Item registry service because the role registry service depends on the Item registry service and the Item registry service depends on the role registry service. These two services are in the bundle **org.openhab.core**, so the bundle **org.openhab.core** never goes to the ACTIVE state which is one of the main bundles of openHAB where many bundles depend on it. As a result, the running instance of openHAB is never complete. For this problem, a question has been asked in the openHAB community [52]. To solve this problem, the role registry will not use or depend on the service Item registry. Instead, the information about an Item that the service role registry needs will be placed in the arguments to its functions. Since the service role registry does not depend on the service item registry, there is no longer a service deadlock for the service role registry and for the service item registry. The bundle **org.openhab.core** is able to switch to the ACTIVE state. As a result, the openHAB instance is working properly.

4.8.5 Cache problems with Maven

From time to time, the openhab-core project has been updated to take into account changes made by other openHAB contributors. Following an update, the openhab-core project will not be built correctly with *Maven*. This is due to the existence of a local *Maven* cache. During the build of the updated project, *Maven* will use information in its local cache that is completely irrelevant to the updated project. This irrelevant information in the local cache of *Maven* will result in errors when building the project openhab-core. As a result, the build of the openhab-core project with *Maven* will fail. For this problem, a post has been requested from the openHAB community [63]. To solve this problem, the local cache of the computer must be deleted. To remove the cache of the Linux operating system, the directory of this location `/home/<User_Name>/.m2` must be deleted, for other operating systems, the location is specified in the support section of the CloudBees forum [17]. Once the *Maven* cache is deleted, the updated version of the openhab-core project without any modifications must be built to put the correct data in the local *Maven* cache. As a result, the openhab-core project with modifications can be built and run in the openHAB distribution successfully.

4.8.6 JSON data management

As explained in the Section 4.6.2 the data that contains RBAC information for openHAB users must be exchanged between the client and server side. The data is in JSON format. Therefore, on the server side, functions must be implemented to transform the

[12] The role registry is inside the bundle **org.openhab.core** and defined with the interface `RoleRegistry`, see Section 4.4

[13] The item registry is inside the bundle **org.openhab.core** and defined with the interface `ItemRegistry`

data from Java object format to JSON format and from JSON format to Java object format. These functions are implemented in the **org.openhab.core.io.rest.core** bundle in the `AccessControlResource`, `AccessControl` and `UserAccessControl` classes[14]. These implemented functions allow the server side of openHAB to process the received data (containing information about the RBAC model) in JSON format and send the necessary information about the RBAC model to the client side in JSON format.

4.8.7 Binding installation

To be able to install the Bindings from the scenario (see Section 3.3.3) in the running instance of openHAB that contains the RBAC model, they must be installed before a bundle is replaced in the openHAB distribution (replacing a bundle in the running instance of openHAB is explained in Section 4.3.4). Otherwise (if a bundle has already been replaced in the running instance of openHAB), it will not be possible to install the bindings. The scenario will not be configured for the running instance that contains the RBAC template. as a result, this is a problem because issues related to access control management in the scenario will not be resolved. To solve this problem, the scenario must first be configured in the current instance of openHAB and then the modified bundles can be put into the current instance of openHAB.

4.8.8 Lack of resources

To implement the interaction between the client-side and the server-side the openhab-core project (server-side) and the openhab-webui project (client-side) must be modified. In addition to being modified, they must be run at the same time to see the modification on the main UI (see Section 4.3.3) and on openHAB (server-side). The meaning that both projects must be run at the same time is that the steps explained in Sections 4.3.1, 4.3.2, 4.3.3 and 4.3.4 to run the openHAB distribution, debug the openHAB distribution, see the changes made to the main interface, and incorporate new bundles into the running instance of openHAB must be done at the same time. Therefore, this requires a lot of resources.

In the first case, the project openhab-webui was opened with the VSCode editor to have the Vue.js extension. It was not open with Intellij because Vue.js files in IntelliJ can only be managed with IntelliJ IDEA Ultimate which is a paid version of the IntelliJ IDE. So by running the openHAB instance (see Section 4.3.1), then debugging it (see Section 4.3.2), opening the openhab-webui with VSCode, then running it with `npm start` (see Section 4.3.3) and building one of the two projects openhab-webui or openhab-core with *Maven*. (see Section 4.3.4) at the same time will crash the computer every time. The computer did not have enough resources to perform so many tasks.

The first attempt to overcome this lack of resources was to use a Raspberry Pi. The idea was to do the steps to run the openhab-core project on the computer and the steps

[14]The classes `AccessControlResource AccessControl` and `UserAccessControl` are at the location `<openhab-core project>/bundles/org.openhab.core.io.rest.core /src/main/java/org/openhab/core/io/rest/core/internal/accessControl`

to run the openhab-webui project on the Raspberry Pi to split the required resources in two. The steps for running the openhab-core project on the computer are explained in the Section 4.3.1. The steps for running the openhab-webui project on the Raspberry Pi are the same as for the computer (see Section 4.3). However, for the latter, there are a few additional steps to perform as follows:

- The commands for the steps explained in the Section 4.3.3 must be executed in a remote terminal connected to the Raspberry Pi with SSH (see Section 3.4).
- The openHABian process running on the Raspberry Pi must be killed (see Section 4.7).
- Instead of executing the `npm start` command the `OH_APIBASE=http://<TailScaleIpAddressOfTheComputer>:8080/ npm start` command must be executed because the running instance of openHAB is on the computer and not on the localhost of the Raspberry Pi.
- More RAM on the Raspberry Pi has been allocated for Node.js to avoid the Raspberry Pi crashing. To know the available RAM allocated to Node.js, the command `node -e 'console.log(v8.getHeapStatistics().heap_size_limit/(1024*1024))'` has been executed. The RAM[15] that has been allocated to Node.js on the Raspberry Pi was 900MB with the command `export NODE_OPTIONS="-max-old-space-size=900"`.
- Ngrok has been used to create an HTTP and HTTPS tunnel to forward traffic running on the port 8081 of the Raspberry Pi to a remote server with the command `./ngrok http localhost:8081 -host-header="localhost:8081"` [8]. Ngrok generates an address that can be reached anywhere, which corresponds to the address `localhost:8081`. Therefore, the main UI with the changes can be reached at the address generated by ngrok with the computer.
- Connection has been realized to the Raspberry Py file system to edit the project openhab-webui and thus modify, manage and change the main UI of openHAB (see Section 4.3.3).

Finally, the Raspberry Pi was not the right solution to solve the resource problems because it crashes very often, reaching the main interface it is very slow because of ngrok which transmits the packets of the running instance of openHAB by a remote server, and the modifications made to the main interface are taken into account slowly.

The second solution was to use a remote server to have enough resources. With the approval of Professor Etienne Riviere, a remote virtual machine (VM) running on the INGI cloud of UCL was obtained. For security reasons, the steps to access the remote VM running on the INGI cloud will not be explained. What was planned to be done on the remote VM running on the INGI cloud was to build the openhab-core project and the openhab-webui project with *Maven* which requires and takes a lot of resources. Although

[15] *Random access memory (RAM) is a computer's short-term memory, where data that the processor is currently using is stored temporarily. RAM memory can be accessed much faster than data on a hard disk, SDD, or another long-term storage device, which is why RAM capacity is so important for system performance [6]*

the VM is powerful enough (2 virtual CPUs and 8 GB of RAM), its file system directory with the most persistent memory is only 4071 MB. The openhab-core project and the openhab-webui project together has and need 2.1 GB (taking into account the files created by *Maven*) of persistent memory. Therefore, the amount of memory in the remote VM was not enough to build the openhab-core project or the openhab-webui project with *Maven*. In conclusion, the remote VM was not the solution to the lack of resources.

The final solution for the lack of resources is to use IntelliJ IDEA Ultimate which is a paid version of IntelliJ IDEA. The paid version of IntelliJ IDEA Ultimate allows optimal management of Vue.js files. The problem encountered since the beginning was due to the use of the VSCode IDE with the Vue.js extension for the openhab-webui project. The openhab-webui project has a huge amount of Vue.js files. VSCode (which uses the Vue.js extension) is not at all optimized to handle Vue.js files and thus uses a huge amount of resources. Because of using VSCode (using the Vue.js extension) to handle the openhab-webui project the computer crashed every time. When IntelliJ IDEA Ultimate was used, the management of Vue.js files was very optimized, therefore the computer crashed from time to time, but very rarely. With IntelliJ IDEA Ultimate, it will be possible to manage, build and run both projects: openhab-webui and openhab-core. To conclude the problem of lack of resources has been solved.

4.9 Potential security vulnerabilities of the openHAB IoT ecosystem

The various challenges that had to be overcome to be able to configure the scenario in openHAB with fine-grained access control for the platform users through the RBAC model were described. As mentioned in the introduction (see Section 2), access control is one of the main features to ensure good security in an IoT ecosystem like a smart home configured with openHAB. Therefore, security vulnerabilities found in openHAB will be stated and an explanation will be given if they have been fixed or not.

Information about users, roles, and groups is stored in files in openHAB as explained in the Section 4.4. These files, therefore, contain critical information about the platform as they are used to manage access control for openHAB with the RBAC model. To ensure that no one modifies these files, each time these files are accessed by openHAB, the integrity of these files must be checked. If the integrity is corrupted, the files should not be valid and the data that the file contains should be generated again. The integrity of a file can be verified by using cryptographic algorithms to generate a digital signature for the file that can be verified afterward (such as JWTs as explained in the Section 4.1.3). However, this is not really a security vulnerability because the malicious person who wants to modify sensitive files must first obtain superuser rights on the machine where openHAB is hosted. Getting superuser rights to the machine is highly complicated. Therefore, if an attacker succeeds in obtaining superuser rights, it means that the attacker has already succeeded in exploiting the machine and thus openHAB. A post was created on the openHAB community [55], which sparked a debate about whether this was really a security flaw or not. The conclusion of the debate is that it is not, but that it is better

and safer to check the integrity of the files, regardless of the rights required to modify the files. Due to time constraints, this feature has not been implemented in openHAB.

As already mentioned in the Section 4.6.1, the command to delete a user or create a user can be executed by any user with access to the smart home's openHAB console. So assuming that a user with the least privilege in the platform or even a malicious person knows the openHAB console password (by default it is *habopen*[16]) and successfully reach it. He will be able to delete users with the role *administrator* and create a new user with the role *administrator* whose password he knows. Users are deleted and created with the functions of the `openhab:users` command (see Section 2.5.1). As a result, the simple user or the malicious person removes all the access of the users who had the highest privilege and gets all the access to the platform. This security flaw has been fixed by requesting the password and username of a platform user with the role *administrator* (see Section 4.6.1). In addition, a check that the parse string (which contains the username and password) contains only alphabetical numbers and letters should be done to add protection against the Buffer Overflow attack.

The purpose of JWTs and their management in the openHAB platform has been described in detail in Section 4.1.3. Figure 4.4 explains how the management of JWTs in openHAB is handled. However, all the features and steps explained in the Figure 4.4 to properly integrate JWTs management were not well implemented in openHAB. Indeed, there are three steps in the Figure 4.4 that are not well implemented and therefore lead to security vulnerabilities.

The first is in step 1 of Figure 4.4. In step 1, the password sent by the user to the platform is not hashed. Therefore, a person can scan the smart home's local network and retrieve the password of the user who may be a user with the role of *administrator*. To prove this security vulnerability, the local network where an instance of openHAB is running was scanned with Wireshark and as shown in the Figure 5.9 the password of the user *nico* can be obtained. To solve this security problem, the password must be hashed on the client-side and the server-side must store the hashed password and verify the password with the hashes. This security problem has not been fixed in openHAB.

The second is in step 7 of Figure 4.4 where the JWT must be checked for each request made by a user of openHAB. In openHAB, a user's JWT is checked only when the client-side page is refreshed. Therefore, anyone can forge an HTTP request and send it to the server to get information about the smart home for example. The server will accept the HTTP request because it does not check the JWT in the HTTP request header. This security vulnerability has been fixed for HTTP methods that are used and managed to handle RBAC for openHAB users but have not been fixed for all HTTP methods used and managed by openHAB.

The third step is step 9 in Figure 4.4. In step 9 the user (client-side) has to verify the JWT put in the HTTP response with the RSA public key that matches the RSA private

[16]The password of the openHAB console can be changed. How to change it is explained in the openHAB documentation [60].

key used by openHAB (server-side) to generate the JWTs. With this verification, the user (client-side) can be sure that the HTTP response is coming from openHAB (server-side). However, the client does not receive the RSA public key to verify the JWT in the HTTP response header. Therefore, anyone can forge and send a fake HTTP response to a user (client-side) and the user will accept the fake HTTP response because the JWT in the HTTP response header is not verified. To solve this security problem, the user (client-side) must receive the RSA public key used by openHAB (server-side) to generate the JWT. Then, the user (client-side) must verify the JWT in every HTTP response received by openHAB (server-side). This security issue has not been fixed in openHAB.

A malicious person is able to exploit any of the above security vulnerabilities only if he has access to the local network where openHAB is running. Considering that the local network is a network with the minimum required security. In order to access the local network, the user must know the Wi-Fi password of the local network. Furthermore, the local network is protected by a firewall. In addition, HTTPS can be used in openHAB (at port 8443). Thus, in case of HTTPS is used, (for the security vulnerability in step 1 of Figure 4.4) the password cannot be recovered but it is still preferable to send and process passwords that are hashed, and (for the security vulnerability in step 7 of Figure 4.4) the client side will not accept false incoming HTTP responses due to the HTTPS connection. However, even with HTTPS, the vulnerability in step 9 of Figure 4.4 can be realized because the server accepts HTTP requests. Fortunately, the JWT in the HTTP request header for critical resources of the openHAB instance with the RBAC model is checked. Therefore, openHAB with the RBAC model embedded does not accept false HTTP requests for critical resources (as the HTTP request to put/get RBAC information to/from openHAB). In the documentation of openHAB [60], it is stated that it is vitally important that the openHAB instance is not exposed to the Internet because there is no authentication in place for the users and surely because of the potential security vulnerability found.

To conclude, security problems have been solved in openHAB. By requesting the username and password of a user with the role of *administrator* when sensitive commands in the openHAB console are executed. The JWT is checked by openHAB (server-side) for HTTP requests made by openHAB users (only for HTTP methods that request critical resources). Finally, the vulnerabilities in steps 1 and 9 of Figure 4.4 can be resolved using HTTPS (with port 8443). A post has been created on the openHAB community [65], explaining how the RBAC model was added to openHAB and the potential security vulnerabilities explained above. In addition to the explanation of how the RBAC model was incorporated into openHAB for the created post on the openHAB community [65], there is the link of a video [28] that explains the configured scenario (see Section 2.5.2), why the RBAC model is useful for the scenario, and how RBAC for openHAB users is configured for the scenario.

Chapter 5

Conclusion

The thesis shows why openHAB is an efficient, powerful, and robust open-source platform for managing the IoT ecosystem of a smart home. OpenHAB won many awards. A comparison between open-source home automation platforms and proprietary IoT platforms is described. A study ranked openHAB second among twenty existing open-source platforms for IoT devices [76]. However, it is demonstrated through a scenario that openHAB has almost no access control for its users. In the scenario, which is a smart home with a family, the access control to the IoT ecosystem devices must be customized for each user (child). To be able to customize the access control for the users in the scenario, the RBAC model is chosen. It is chosen because the RBAC model fits well with the requirements of the missing features for access control management for openHAB users.

After choosing the RBAC model in openHAB, the concepts and terms of openHAB like the semantic model, Item syntax, Bindings, Things, Channels, and Items are explained to be able to configure the scenario that needs the RBAC model for its users. The steps to configure the scenario (semantic model and Item syntax respected, Bindings installed, Things, Channels, Links, and Items configured) are detailed to be apt to set up the scenario in a running instance of openHAB.

The technical part of openHAB is illustrated, exposed, and clarified and the steps to manage and run an instance of openHAB are explained to understand how RBAC for openHAB users can be implemented in the platform. With the understanding of the home automation software architecture and the managing of a running instance of openHAB, RBAC for openHAB users is successfully incorporated into the technology-agnostic home automation platform. The RBAC model in the IoT platform enables fine-grained access control for openHAB users in the IoT ecosystem of a domotic. Users have groups of roles and roles defining the resources they can access as well as the resulting responsibilities from that access. The management of the roles and groups is feasible with the openHAB console and the main UI. As a result, the lack of access control for the scenario is solved, fulfilled, and satisfied because each child of the smart home will be capable of only accessing its own devices.

Enabling RBAC for openHAB users was not an easy task. Many challenges are encountered. The challenges are to be capable to add an external package/library into an

openHAB bundle, resolving errors with the Apache Karaf framework, handling classes outside and inside the internal package of a bundle, avoiding service deadlock, putting correct data in the local *Maven* cache, manipulate JSON object, correctly set up the Bindings to the running instance of openHAB where the RBAC model is embedded, and have enough resources to build and run the openhab-webui and the openhab-core project.

By implementing RBAC for openHAB users for a running instance of openHAB, potential security vulnerabilities are found. Authentication is not required for sensitive commands in openHAB consoles. OpenHAB does not hash the transferred password. On the client-side, there is no public key to verify the signature of the openHAB server and thus the incoming HTTP method. Finally, openHAB does not handle user authentication for each HTTP request. Therefore, openHAB practitioners must absolutely use HTTPS to encrypt the transferred password and to authenticate every HTTP response sent on the client-side. In addition, it is vitally important that the running instance of openHAB remain on a local network protected by a firewall that requires a password to log in. Fortunately, sensitive openHAB commands and critical HTTP requests on the server-side are authenticated for the openHAB instance that contains the RBAC model.

RBAC for openHAB users is fully integrated into openHAB. However, the part of the main UI that handles RBAC for openHAB users should be improved to properly handle the RBAC model in a user-friendly and understandable way. HTTP requests from openHAB on the server-side are authenticated and the resources returned to the request are based on the roles and groups assigned to users only for critical openHAB resources. Therefore, this should be implemented for all home automation software resources to enforce the RBAC model for the entire technology-agnostic home automation platform.

A home automation platform needs strong security because it contains highly sensitive data about the users of a domotic. For this reason, the security of a smart home environment is fundamental and should not be overlooked. Therefore, by enabling RBAC for openHAB users the security of the openHAB IoT ecosystem increases remarkably by authenticating every request made to the platform with groups and roles assigned to users. RBAC for openHAB users is really important, essential, and crucial to manage resource access for users and to ensure a high level of security for the openHAB IoT ecosystem.

Appendix

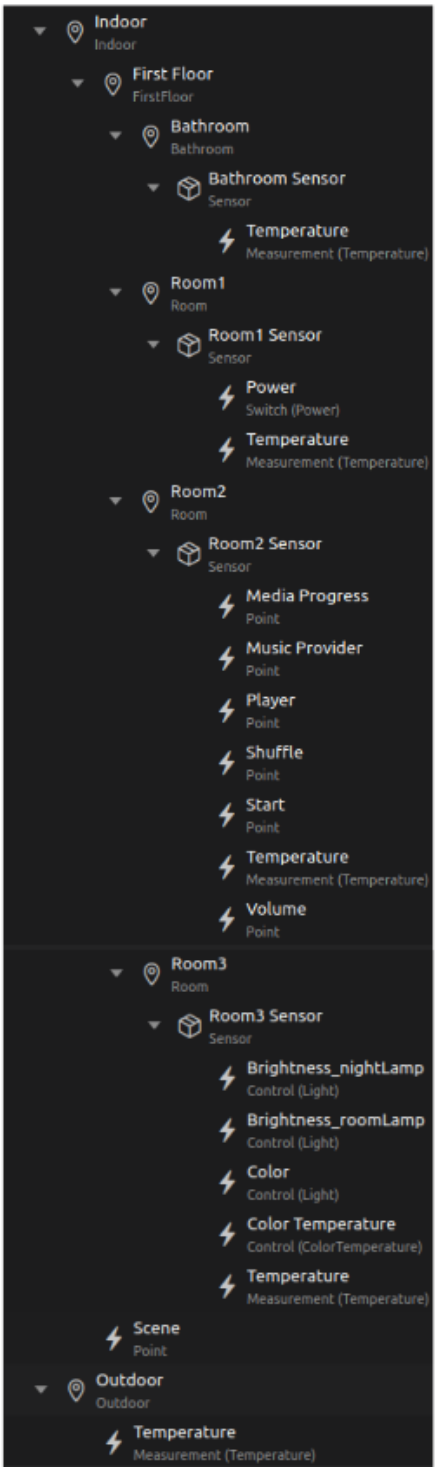

Figure 5.1: Tree view of the semantic model of the scenario

Figure 5.2: Access for the administrator user

Figure 5.3: Access for the child1 user

Figure 5.4: Access for the child2 user

Figure 5.5: Access for the child3 user

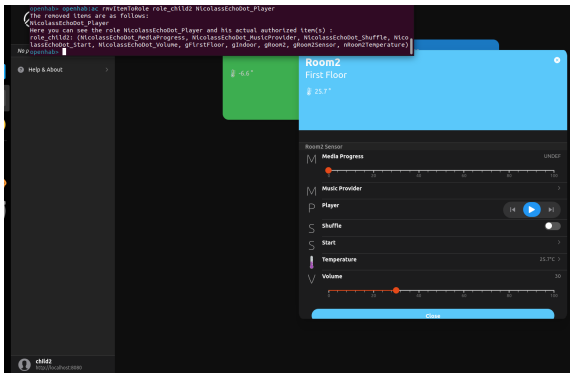

Figure 5.6: Deletion of an Item for the child2 user

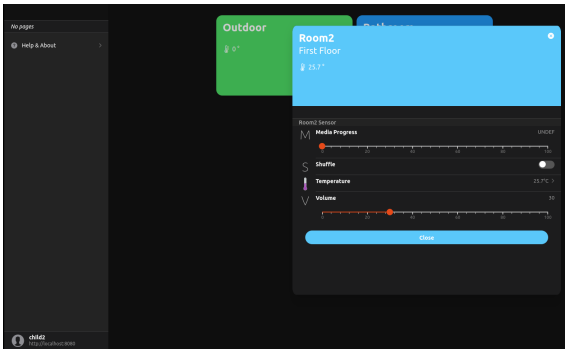

Figure 5.7: Suppression of Items **NicolassEchoDot_Player**, **NicolassEchoDot_MusicProvider** and **NicolassEchoDot_Start** for the child2 user

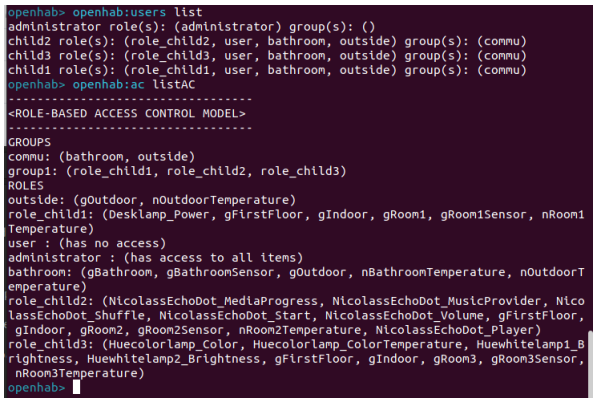

Figure 5.8: Configuration of the RBAC for the scenario

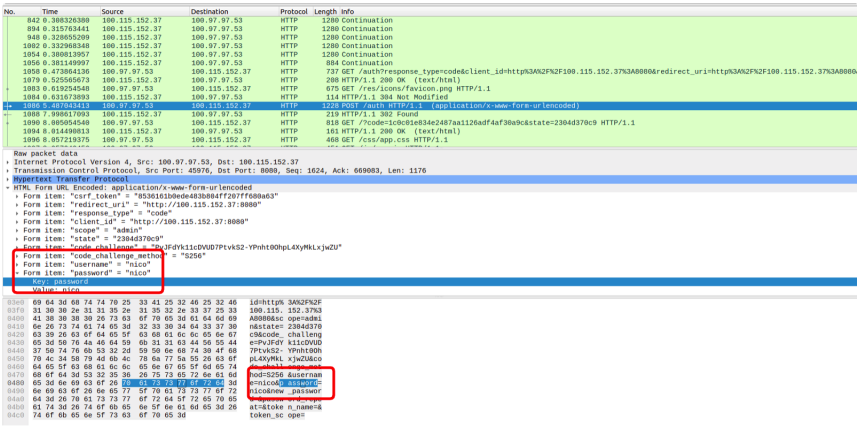

Figure 5.9: Password recovery of a user with Wireshark

Bibliography

[1] Allen, J., and Collison, S. Black duck open hub created 1 january 2006. <https://www.openhub.net/>. accessed 12 May 2022.

[2] Amazon. Aws iot device management pricing. <https://aws.amazon.com/iot-device-management/pricing/>. accessed 19 May 2022.

[3] Apache. Kar. <https://karaf.apache.org/manual/latest/kar>. 5 June 2022.

[4] auth0. Introduction to json web tokens. <https://jwt.io/introduction/>. 20 Mai 2022.

[5] authO. Debugger. <https://jwt.io/>. 20 Mai 2022.

[6] AvastAcademy. Wath is ram? <https://www.avast.com/>. 5 June 2022.

[7] Awards, I. Honoring the year's best products, organizations and ideas shaping the iot. <https://web.archive.org/web/20160306190817/http://postscapes.com/internet-of-things-award/2014/winners.html>. accessed 19 April 2022.

[8] banjocode. How to allow remote access to your localhost apps with ngrok, 20 march 2021. <https://www.banjocode.com/post/tools/>. 1 June 2022.

[9] Barker, S., and Rothmuller, M. *The Internet of Things—Consumer, Industrial and Public Services 2016–2021.* Juniper Research, 1 April 2020. accessed 19 May 2022.

[10] Bhuwad, A. Best guide to json web token (jwt), 28 november 2020. <https://medium.com/swlh/>. 20 Mai 2022.

[11] BlackDuckOpenHub. openhab - empowering the smart home. <https://www.openhub.net/p/openHAB>. accessed 10 april 2022.

[12] Brice, A. Best of open source smart home: Home assistant vs openhab. <https://smarthome.university/>. accessed 15 May 2022.

[13] Cambose, A. How jwt works — in depth, 4 jun 2020. <https://medium.com/swlh/how-jwt-works-in-depth-604c93ec20a4>. 20 Mai 2022.

[14] Carey, L. Jax innovation awards 2014 spotlight: openhab. <https://jaxenter.com/jax-innovation-awards-2014-spotlight-openHAB-107776.html>. accessed 8 May 2014.

[15] CeHu, V., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., and Scarfone, K. *Guide to attribute based access control (ABAC) definition and considerations*. NIST Special Publication 800-162, January 2014.

[16] CeHu, V., Kuhn, R., and Yaga, D. *Verification and Test Methods for Access Control Policies/Models*. NIST Special Publication 800-192, June 2017.

[17] Cloudbees. How to clear maven cache, 18 january 2021. <https://support.cloudbees.com/hc/en-us/articles/>. 5 June 2022.

[18] Danny. The ultimate smartthings to home assistant migration guide, 18 january 2021. <https://smarthomepursuits.com/>. accessed 19 May 2022.

[19] DeNardis, L., Mohammadpour, A., Caso, G., Ali, U., and DiBenedetto, M.-G. *Internet of Things Platforms for Academic Research and Development: A Critical Review*. Applied Sciences, February 2022.

[20] Eclipse. Eclipse foundation. <https://www.eclipse.org/membership/showMember.php?member_id=1209>. accessed 10 april 2022.

[21] Eclipse. Eclipse membership. <https://www.eclipse.org/membership/#tab-benefits>. accessed 10 April 2022.

[22] Eclipse. Equinox osgi. <https://www.eclipse.org/equinox//>. accessed 15 May 2022.

[23] Etma. What is the internet of things ecosystem? <https://www.etma.org/what-is-iot-ecosystem/>. accessed 19 May 2022.

[24] Fingas, R. Samsung smartthings: Your guide to samsung's smart home platform, 10 february 2022. <https://www.androidauthority.com/samsung-smartthings-guide-3049705/>. accessed 19 May 2022.

[25] FortuneBusinessInsights. Iot market size, share, growth, trends, business opportunities, iot companies, statistics, report 2028. created 10 november 2021. <https://www.globenewswire.com/news-release/2021/11/10/2331267/0/en/>. accessed 19 May 2022.

[26] FreeBSD. Ssh man page. <https://manpage.me/?q=ssh>. 5 June 2022.

[27] Gennart, N. Manage user roles in the karaf console #2765. <https://github.com/openhab/openhab-core/pull/2765>. 5 June 2022.

[28] Gennart, N. Role-based access control for openhab users. <https://www.youtube.com/watch?v=7cZNDdOb9pU>. 5 June 2022.

[29] Github. openhab fails to start if addons.kar is used, 15 august 2017. <https://github.com/openhab/openhab-distro/issues/519>. 5 June 2022.

[30] Google. Google cloud iot core pricing. <https://cloud.google.com/iot/pricing>. accessed 19 May 2022.

[31] Google. Material design litecreated june 25, 2014. <https://getmdl.io/index.html>. accessed 13 May 2022.

[32] Google. What it can do. <https://assistant.google.com/services/a/uid/000000f5c61c627e?hl=en-US&source=web>. accessed 15 May 2022.

[33] Gupta, M., Bhatt, S., Alshehri, A. H., and Sandhu, R. *Access Control Models and Architectures For IoT and Cyber Physical Systems*. Springer, January 2022.

[34] Heimgaertner, F., Hettich, S., Kohlbacher, O., and Menth, M. *Scaling home automation to public buildings: A distributed multiuser setup for openHAB 2*. Global Internet of Things Summit (GIoTS), 2017.

[35] Hussein's, A. H. *Internet of Things (IOT): Research Challenges and Future Applications*, vol. 10. (IJACSA) International Journal of Advanced Computer Science and Applications, 6 November 2019.

[36] Jetty. The eclipse jetty project. <https://www.eclipse.org/jetty/>. accessed 11 April 2022.

[37] Johnson, W. Rs256 vs hs256 what's the difference? 4 mai 2022. <https://auth0.com/blog/rs256-vs-hs256-whats-the-difference/>. 20 Mai 2022.

[38] KARAF. The modulith runtime. <https://karaf.apache.org/>. accessed 10 April 2022.

[39] Krishna, G., and Myadam, R. Why open source tools are popular for developing an iot ecosystem, on 10 march 2021. <https://www.opensourceforu.com/2021/03/why-open-source-tools-are-popular-for-developing-an-iot-ecosystem/>. accessed 20 April 2022.

[40] Langaliya, C., and Aluvalu, R. *Enhancing Cloud Security through Access Control Models: A Survey*. International Journal of Computer Applications, 7 February 2015.

[41] Lazakidou, A., Siassiakos, K., and Ioannou, K. *Wireless Technologies for Ambient Assisted Living and Healthcare: Systems and Applications*. Medican Information Science ReferenceMedican Information Science Reference, January 2010.

[42] Lee, M. Smartthings vs openhab (differences between smartthings and openhab), 16 november 2020. <https://www.diysmarthomehub.com/>. accessed 19 May 2022.

[43] Lutolf, R. Smart home concept and the integration of energy meters into a home based system. In *Proc. 7th Int. Conf. Metering Apparatus and Tariffs for Electricity Supply* (6 November 1992), pp. 277–278.

[44] MarketsandMarkets. Iot technology market with covid-19 impact analysis, by node component (sensor, memory device, connectivity ic), solution (remote monitoring, data management), platform, service, end-use application, geography—global forecast to 2027. 2021. <https://www.marketsandmarkets.com/Market-Reports/iot-application-technology-market-258239167.html>. accessed 19 May 2022.

[45] Maven. Welcome to apache maven, last published 22 may 2022. <https://maven.apache.org/index.html>. 26 Mai 2015.

[46] MDN. Http request methods. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>. 20 Mai 2022.

[47] Miraz, M. H., Ali, M., Excell, P. S., and Picking, R. *A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT).* Conference: 2015 Internet Technologies and Applications (ITA), september 2015.

[48] moo. Moo! <https://www.npmjs.com/package/moo>. 1 June 2022.

[49] nearley.js. Getting started. <https://nearley.js.org/docs>. 1 June 2022.

[50] openHAB. Add/remove user roles via karaf console #2453. <https://github.com/openhab/openhab-core/issues/2453>. 5 June 2022.

[51] openHAB. Download openhab with openhabian. <https://www.openHAB.org/download/>. accessed 17 May 2022.

[52] openHAB. How to use the roleregistry service in its own bundle? (i have implemented the roleregistry service), 30 march 2022. <https://community.openhab.org/t/>. 5 June 2022.

[53] openHAB. Howto: Setup intellij for binding development. <https://community.openhab.org/t/howto-setup-intellij-for-binding-development/87135>. 5 June 2022.

[54] openHAB. Interface chart. <https://www.openHAB.org/javadoc/latest/org/openHAB/core/model/sitemap/sitemap/chart>. accessed 11 April 2022.

[55] openHAB. Manage storage of user credentials, 25 january 2022. <https://community.openhab.org/t/manage-storage-of-user-credentials/132362/>. 5 June 2022.

[56] openHAB. openhab add-on reference. <https://www.openHAB.org/addons/>. accessed 5 April 2022.

[57] openHAB. openhab community. <https://community.openHAB.org/>. accessed 14 April 2022.

[58] openHAB. openhab documenation - contribution. <https://www.openHAB.org/docs/developer/contributing.html>. accessed 12 April 2022.

[59] openHAB. openhab documentation - coding guidelines. <https://www.openHAB.org/docs/developer/guidelines.html>. accessed 12 April 2022.

[60] openHAB. openhab documentation updated 22 december 2020. <https://www.openHAB.org/docs/>. accessed 5 April 2022.

[61] openHAB. openhab github repositories. <https://github.com/openHAB>. accessed 12 April 2022.

[62] openHAB. openhabian hassle-free openhab setup. <https://www.openHAB.org/docs/installation/openHABian.html>. accessed 18 April 2022.

[63] openHAB. Problem when i build the openhab-distro project and when i run openhab to test it, 28 march 2022. <https://community.openhab.org/t/>. 5 June 2022.

[64] openHAB. Problem with the openhab-distro project, 7 february 2022. <https://community.openhab.org/t/problem-with-the-openhab-distro-project/133012>. 5 June 2022.

[65] openHAB. Rbac model in openhab (and potential security vulnerability found), 4 june 2022. <<https://community.openhab.org/t>. 5 June 2022.

[66] openHAB. Release 3.2.0, 19 december 2021. <https://www.openHAB.org/addons/>. accessed 5 April 2022.

[67] openHAB. Release 3.3.0, 3 april 2022. <https://www.openHAB.org/addons/>. accessed 5 April 2022.

[68] openHAB. [solved] framework event error after recent update, 17 august 2017. <https://community.openhab.org/t/>. 5 June 2022.

[69] openHAB. Summary. <https://community.openhab.org/u/ngennart/summary>. 5 June 2022.

[70] OSGi. What is osgi? <https://www.osgi.org/resources/what-is-osgi/>. accessed 10 April 2022.

[71] OSGiAlliance. *OSGi Service Platform Core Specification*. OSGi Alliance, Release 4, Version 4.2, June 2019.

[72] Raisul, M., Bin, M., and Alauddin, M. *A Review of Smart Homes – Past, Present, and Future*. IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews), November 2012.

[73] repository, M. Apache karaf :: Deployer :: Karaf archive (.kar) » 4.3.6. <https://mvnrepository.com/artifact/org.apache.karaf.deployer/org.apache.karaf.deployer.kar/4.3.6>. 29 Mai 2022.

[74] repository, M. Gson. <https://mvnrepository.com/artifact/com.google.code.gson/gson>. 29 Mai 2022.

[75] repository, M. Osgi service component annotations. <https://mvnrepository.com/artifact/org.osgi/org.osgi.service.component.annotations>. 29 Mai 2022.

[76] Setz, B., Graef, S., Ivanova, D., Tiessen, A., and Aiello, M. *A Comparison of Open-Source Home Automation Systems*. IEEE Access, Decembre 2021.

[77] Singhal, A., Winograd, T., and Scarfone, K. *Guide to Secure Web Services*. NIST Special Publication 800-95, August 2007.

[78] Spotify. Spotify on alexa devices. <https://support.spotify.com/us/article/spotify-on-alexa-devices/>. 5 June 2022.

[79] Tailscale. A secure network that just works. <https://tailscale.com/?msclkid=ca160fca065c15af76506d9795e379f1>. accessed 18 April 2022.

[80] TechTerms.com. Http definition. <https://techterms.com/definition/http>. 25 Mai 2015.

[81] Torvalds, L. git. <https://git-scm.com/>. 29 Mai 2022.

[82] Wikipedia. Facebook–cambridge analytica data scandal. <https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal>. accessed 19 May 2022.

[83] Wikipedia. Jar (file format). <https://en.wikipedia.org/wiki/JAR_(file_format)>. 29 Mai 2022.

[84] Wikipedia. localhost. <https://en.wikipedia.org/wiki/Localhost>. 29 Mai 2022.

[85] Wikipedia. openhab. <https://en.wikipedia.org/wiki/openHAB>. accessed 19 April 2022.

[86] Wikipedia. Smart home. <https://de.wikipedia.org/wiki/Smart_Home>. accessed 21 May 2022.

[87] Wschool. What is the rest resources? <https://www.w3schools.in/restful-web-services/rest-resources>. 31 Mai 2022.

[88] Xtext. Language engineering for everyone! <https://www.eclipse.org/Xtext/>. accessed 13 April 2022.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**