



Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Networked Systems and Services

Integrating open-source smart home software into a physical model

BACHELOR'S THESIS

Author

Sándor Bence Gombos

Advisor

Róbert Schulcz

December 3, 2024

Contents

Kivonat	i
Abstract	iii
1 Introduction	1
1.1 Motivation, previous work	2
1.2 A brief history of smart home systems, protocols	3
1.2.1 Protocols	3
1.2.2 Solutions	5
1.3 Architecture and devices of a typical smart home system	6
2 Planning	9
2.1 General goals, requirements and principals of the planned model	9
2.2 Hardware environment	9
2.3 Software environment	11
3 Implementation	13
3.1 Floor plan	13
3.2 Connecting devices to the microcontroller	13
3.3 Network setup	16
3.4 Software setup	17
3.4.1 Home Assistant initial setup, migration	17
3.4.2 ESPHome	19
3.4.3 Additional user interface setup, customization	21
3.4.3.1 Entity customization	21
3.4.3.2 Dashboard customization	22
3.4.4 Smartphone setup	24
3.4.5 Automations	26
3.4.6 Voice assistant	28

4	Review	29
4.1	Usage	29
4.2	Critical analysis, shortcomings	30
4.3	Opportunities of further development	31
	Acknowledgements	33
	Bibliography	38

HALLGATÓI NYILATKOZAT / STUDENT DECLARATION

Alulírott *Gombos Sándor Bence*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

I, *Sándor Bence Gombos* hereby declare, that I have made this thesis myself, without the usage of unpermitted help and have only used sources (i.e. professional literature, tools etc.) noted in the bibliography. For every part, which I have used word by word or with the same meaning, but in different words, I made so with the clear and obvious marking of the source.

I agree, that BME VIK may publish my work's metainformation (author(s), title, English and Hungarian abstract, year of publication, name(s) of advisor(s)) in a publicly available, electronic manner and publish the full text content via the University's internal network (or for authenticated users). I declare, that the work turned in and its electronic version are the same. In the case of diploma thesis designs encrypted by the dean's permit, the content of the thesis may only be available 3 years later.

Budapest, 2024. december 3.

Gombos Sándor Bence
hallgató / student

Kivonat

Az okosotthon rendszerek és otthon automatizációk technológia alapú megoldással nyújtanak kényelmi és egyéb szolgáltatásokat embereknek az otthoni környezetükben. Ez leginkább a ház "okos" eszközökkel (pl. villanykörték, hőmérséklet- és nedvességszenzorok, fűtési- és hűtési vezérlők) való felszerelését jelenti, hálózatra kötve egy vezérlő modullal, amit egy központi felhasználói felületen (általában egy okostelefonnal) lehet irányítani, akár a házról is. Ezek a rendszerek többek is lehetnek, mint egy kényelmi megoldás: növelhetik a ház energiahatékonyságát és biztonságát az erőforrások optimalizálásával, biztonsági felvételek analizálásával és egyéb módokon. Ez a szakdolgozat projekt bevezeti az olvasót az okosotthon rendszerek és otthon automatizálás világába, azoknak történelmébe, általános felépítésébe és kitér a piac vezető gyártóira, azoknak termékeire és megoldásaira. Ezt követően bemutatja egy fizikai demonstrációs modell tervezési és kivitelezési szakaszait, köztük egy hardveres és szoftveres környezet kiválasztását. Végül bemutatja a demo modell használatát, felhasználási módjait és hiányosságait, továbbá annak továbbfejlesztési lehetőségeit.

Abstract

Smart home systems and home automation are technology based solutions for providing convenience and other services for people in their home environments. This is mostly in the form of a house fitted with "smart" devices (eg. light bulbs, temperature and humidity sensors, heating and cooling actuators) networked with a controller module, which can be controlled via a user interface (generally via a smartphone), even remotely outside the house. These systems can be more, than a convenience solution: they can also increase the energy efficiency and safety of the house with resource usage optimization, analysis of security footage and other means. This thesis project introduces the reader to smart home systems and home automation, their history, common architecture and current market leading manufacturers, their products and solutions. It is then followed by a presentation of a physical demo model's planning and implementation phases, including the selection of a hardware and a software environment. Finally, it showcases and evaluates the demo model's general usage, use cases, shortcomings and provides opportunities of further development.

Chapter 1

Introduction

In most conventional, "non-smart" houses or flats, utility and convenience devices (eg. lighting, heating, ventilation) are mostly controlled manually by hand via simple switches, knobs or the combination of the two, which requires people to be physically present at the scene to actuate them. In a home equipped with a smart home system, an extra computer and relay or transistor based system is added to them, which assists their actuation from centralized user interface (most often controlled via a smartphone application or control appliance). Smart home systems can also be extended with digital sensors and other more fine tunable, convenience and security devices (in general just called as "smart" devices), such as temperature and humidity sensors, RGB lights, keycard readers, cameras. The sensors, along with other kinds of input (eg. time of the day, phone sensing) can be used to automate processes, which would otherwise require manual interaction, such as turning on or off lights in rooms based on presence, rolling a shutter on or off at a certain time of a day. The general term for this process is called home automation, but this term also means the process of installation and usage a smart home system, so their meaning is almost identical in most contexts.

I believe that a demo "box" model is an adequate and cost-effective solution to try out the usage of smart home systems and to simulate a proper system installed in a real house. During its implementation phase, I tried to follow two basic principles: cost-effectiveness (cheap price for a thesis project) and the usage of open-source software. We can often hear these days, that companies end supporting and updating their not so old, still completely functional products, which make them vulnerable to attacks and contributes to them becoming hazardous waste (take a look at Amazon Echo Look as an example [13]). This problem can be mitigated to some extent with the usage of actively maintained open source software available to such devices, because this way they can run up-to-date software years after manufacturing, additionally with the devices chosen for the thesis project, they can be later repurposed in other projects (not necessarily home automation related).

The demand for smart home system and home automation technology has been steadily increasing since their inception and there is active market potential in them (the Worldwide Economic Forum predicts it could reach 13 trillion dollars by 2030). Initially, they were considered only as a convenience product (with remote control, a centralized control interface), but since then they have also become a solution of efficiency and safety. Examples for the earlier are automatized heating and motion controlled lights, for the latter are security footage recordings and its analysis, support of handicapped people. [15]

1.1 Motivation, previous work

My motivation behind choosing this thesis project was mostly my previous work on the topic and also that it combines more fields of expertise: various fields in Information Technology along with basic electronics, microcontroller programming.

For Training Project Laboratory, me and my two fellow students used an ESP-based microcontroller to control a few small electronic components connected to it, which in large resembled a house's utilities. However, this was only a very rudimentary breadboard model, with continuous debug printing to the serial console and only a potentiometer and two buttons as inputs, but it was useful to familiarize ourselves with the usage and programming of a modern microcontroller liked by many hobby enthusiasts.

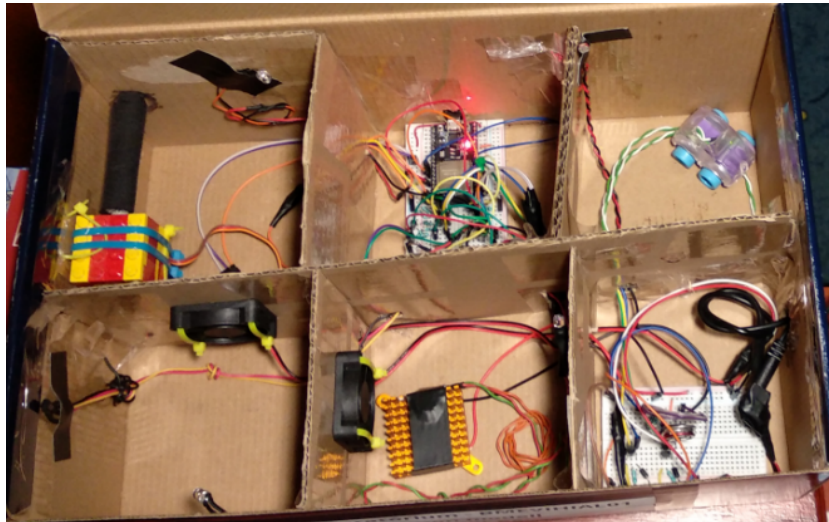


Figure 1.1: Project Laboratory box model contents

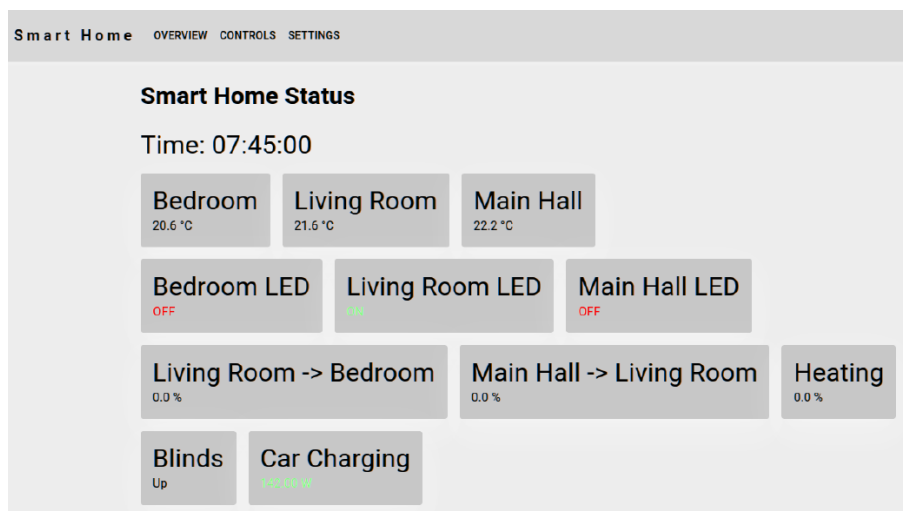


Figure 1.2: Project Laboratory box model user interface

Throughout Project Laboratory, I took the project further and made a proper shoebox smart home system model. Along with a basic floor plan for various rooms and electronic components placed in them (shown in figure 1.1), it featured a web-controllable user interface (shown in figure 1.2). Its software architecture was comprised of three main

components: a JavaScript backend responsible for maintaining the components' status and communication between the microcontroller and user, microcontroller code for controlling the electronic components (reading sensor data, setting output for actuators), receiving and sending data to the backend and finally, a React-based frontend to send commands to and receive readings from the backend.

For the Thesis Project, I've decided to continue work with the shoebox model, but focus more on the software side of improvements - the hardware, electronic components inside the box have remained mostly the same. I believe that the software of my Project Laboratory project was sufficient for showcasing simple smart home use cases in a demo environment, however would have been harder to further develop to extend with more devices and features than using already existing open-source smart home software with good community support, which I had also been looking forward to try out.

1.2 A brief history of smart home systems, protocols

The Third Industrial Revolution's main innovation was the usage of IT and computer technology in order to automate certain industrial processes and achieve a substantial speed up of tasks previously done with manual labor. [48] The appearance of Programmable Logic Controllers (PLCs) and innovation in Robotics made it easier, than ever to precisely control electric devices with a certain logic in industrial installations with much less human supervision. This aspect of computer-based electric and electronic device control is also important for the functioning of smart home systems, however the latter only started to be developed much more later, after a demand appeared for them and manufacturers started to experiment with different technological solutions.

1.2.1 Protocols

The first major breakthrough, that helped the conception of the home automation field was the introduction of the X10 protocol and the devices utilizing it in the 1970s. X10 made it possible to control one lamp and one appliance via a remote control and a central control console by utilizing the house's already existing electrical wiring (on the same phase) to send commands from the control device to the appliances. Later versions of X10-based systems made it possible to control even more devices with additional customization options and the platform still has some popularity among home automation enthusiasts, although it is mainly criticized for its lack of encryption. [14]

However it took a while, before more sophisticated home automation solutions appeared, that utilized computers for controlling the apparatus and interfacing the users. One important aspect of such systems is the way devices communicate to each other, which also required a lot of research and development. After the introduction of wired computer networking protocols, such as Ethernet, TCP/IP, which became the basis for the World Wide Web (WWW) in the early 90s, wireless computer networking, building- and home automation related communication protocols and devices also appeared, but only later in the late 90s. Although many automation devices could be networked via a UTP cable and utilize the aforementioned Ethernet and TCP/IP protocols, the development and usage of wireless protocols made their installation easier and contributed to their spread due to the lower cost of not having to run network cables all around the house besides the power cables and the possibility to make devices portable. IoT, or Internet of Devices is a collective term for the network of physical devices, vehicles, home appliances, and other

embedded devices which connect to each other and exchange data. [42] Home automation devices make heavy use of IoT as well, besides other real-world systems, such as smart grids, autonomous vehicles.

The first version of Wi-Fi was introduced in 1997 and made it possible for devices to communicate wirelessly with a relatively high bandwidth of 2 Mbps on the 2.4GHz ISM band, mainly designed as a Local Area Network (LAN) and Internet access protocol for home and office usage on laptops, PDAs and other mobile computers. Later versions improved bandwidth and some versions utilize the 5 and 6 GHz ISM bands instead for a higher bandwidth connection - but at a lower range. [38]

Z-Wave, introduced in 1999 is a wireless communication protocol designed for residential and commercial building automation and utilizes the 900 MHz range for transmission, therefore achieves higher range at a lower bandwidth, but still sufficient for the targeted kinds of devices, such as garage openers, smoke alarms and motions sensors. [35]

Zigbee, an other low-power communication protocol was introduced in 2003 and originally used the 2.4 GHz ISM band (the 2023 Pro specification added 800-900 MHz support). It is mainly used in home automation, medical data collection, and industrial control systems. [30]

Bluetooth is a short-range wireless protocol introduced in 1997, mainly used to wirelessly connect peripherals to mobile phones, desktops, laptops and point-to-point mobile phone connection. [31] Some of the most common Bluetooth accessories include mice, keyboards, speakers, and headphones. Bluetooth can also be useful in home automation, including appliance control and detecting one's presence with their phone's transmitted Bluetooth signal.

Although wireless communication can be useful for many home automation related use cases, it might not always be the best solution against a wired connection. A solution to ease the burden of wiring is Power over Ethernet (PoE), which was introduced in 2003. [41] Power over Ethernet is a technology in wired Ethernet local area networks, that enables the transmission of power necessary for every operating device to be carried on the same Ethernet cable, that is used for data transmission, therefore eliminates the need of a separate power cord running to the device. Devices, that make active use of it include VoIP phones, video surveillance cameras and high power LED lights. It requires the recipient device to have the necessary power converter electronics and handshake logic, that converts the higher voltage to the required device voltage and the source to be able to power it. If the source device (most likely a switch) it is connected to isn't PoE capable, it can still be made so with the use of a PoE adapter. For PoE, almost always a higher voltage is used than the recipient's internal voltage, because a higher voltage has less resistance for a given amount of power transmitted.

Smart home products really became popular in the 2010s, however with this came a problem among the many products by many manufacturers: they mostly used their own ecosystem, on top of certain standard protocols and this made their cooperation hard, users had to use different interfaces and ecosystems for each owned device. A proposed solution to allow products from different manufacturers to securely communicate in a standardized manner is Matter. [3] The working group behind it was formed in 2019 by Amazon, Apple, Google and other big corporations, and it is being actively developed and adapted by many manufacturers for more and more types of appliances. It is based on the Internet Protocol and uses other existing protocols for the underlying connections, such as Wi-Fi and Bluetooth.

1.2.2 Solutions

In the mid 2000s, companies with a strong focus on home automation products started appearing, such as Control4 in 2004, Insteon and Savant in 2005. [16] [43] [39] Later in the early 2010s, many big IT corporations also jumped onto the hype around smart home systems and released products on the market, often bundled with their voice assistant solutions. This was also helped by the spread of smartphones after the introduction of early iPhone and Android phones and tablets at the end of the earlier decade. Manufacturers could use their software platform to build applications for interfacing users and it also meant that they could make their systems cheaper by only optionally including a control hub device, as that could be substituted by the smartphones or tablets. Such platforms include Amazon's Alexa Smart Home with their Alexa voice assistant and Echo smart speakers, Google's Home with their Nest devices and Assistant voice assistant, Apple's HomeKit software platform. [2] [27] [26] [4] Since then, many home automation, smart home devices and solutions have appeared on the market with a plentiful of competition currently ongoing. [15]

A typical smart home system usually includes smart devices, that mostly provide or control home utilities, along with detectors, sensors and other devices facilitating extra services (eg. voice assistant), additionally a control utility in the form of a dedicated device, or a local or remote computer running a specific control software. Many smart utility devices replace their conventional equivalent (which aren't "smart"), their retrofitting isn't hard in most cases. Such utility devices comprise of controllable monochrome or RGB LED light bulbs and light strips, thermostats for control of heating and cooling (A/C), garage openers, presence detectors, air, humidity and particulate sensors and many more. [51]

In the 2010s, cloud computing technology also started to get traction, as big IT corporations launched theirs, take Amazon Web Services (AWS), Google Cloud Platform (GCP) and Microsoft Azure as examples. Cloud computing means that such providers provide on-demand delivery of IT resources over the Internet, so that that companies can use this to outsource different portions of their infrastructure instead of having to operate everything themselves. [40] It can lead to significant cost savings besides the more agile and faster deployments, scaling of resources according to demand. This technology was also beneficial for smart home solutions, for a number of reasons. [15] Most modern smart home devices require some form of background IT infrastructure (eg. control server) for management, remote access and data collection, which is something that most customers aren't tech savvy enough to set up themselves and operate, regularly update. The amount of data generated also varies device-by-device, typically security cameras would require the most amount of data to be stored somewhere. Because of the many advantages of cloud technologies, manufacturers started to use cloud technologies extensively for their products and focus less on locally hosted solutions.

However, the usage of cloud technology is a double edged sword for a number of reasons. As for concerns against cloud solutions, the first is being vulnerable to the hosting providers and the reliability of their technologies. Although many big IT companies invest heavily in their technologies, downtimes sometimes occur, during which the usage of a smart home system dependent on it might be limited or not possible. An other big concern is privacy and security: devices reside in people's private lives, therefore should be well-protected against access by external parties. A disgruntled hosting or manufacturer employee, state agency, or hackers utilizing a vulnerability or weak passwords might be able gain access to the system and peek into its data, as if the owners were in Big Brother. [36]

Despite the widespread use of cloud-based solutions, many tech-savvy users decide to only use locally hosted smart home platforms with devices either officially or unofficially supporting this way of operation due to the concerns associated with remote- or cloud-based solutions. This way, users have full control of their smart home infrastructure and data, which when set up for remote access, will be as secure as its encryption capabilities and scarcity of vulnerabilities. Such users almost always run open-source smart home software platforms. They also prefer to use hardware made out of open-source platforms, microcontrollers (eg. Arduino, ESP) and basic electronics, or use commercial products with aftermarket open-source firmware flashed onto them instead of utilizing the original software from the manufacturer, which uses remote or cloud services. Home Assistant and OpenHAB are good examples for open-source smart home platforms that can be hosted on a local server and have support for smart many devices from different brands and have good community support from their maintainers and users. [10] [34] ESPHome and Tasmota are two open-source software projects that allow microcontrollers to be easily able to set up in different smart home platforms and control specified electric and electronic devices connected to their pins, eliminating the need to write custom microcontroller code for different platforms, connected components and communication to the control server. [21] [44]

1.3 Architecture and devices of a typical smart home system

Every smart home system solution and product is different, however there are a few key concepts and parts, that most solutions follow and share. In this thesis, a typical IoT-focused smart home system's architecture and devices are presented with some variations, along with the general concepts that they follow. There are also smart home systems which utilize Artificial Intelligence (AI) and its forms in Machine Learning (ML), Deep Learning (DL) and Neural Networks (NN), furthermore Multiagent System (MAS) and Image Processing (IP) based smart home systems, however these aren't going to be covered in the thesis due to them being not used in the accomplished physical demo model presented in the upcoming chapters. [15]

Many IoT-based smart home systems use a controller as a brains of operations, although its form of implementation can vary to a great extent according to the specific platform. One form of implementation is a controller "box" or "hub", where a physical device is present, which manages the appliances, collects data and stores it for later review and provides an interface (on the devices itself, or from a smartphone or computer application) for user control. An other controller solution is a specific control software run on a local or remote (many times cloud-based) computer, to which apparatus connect via an IP-based wired or wireless network solution. A control hub device is sometimes combined with remote or cloud services, therefore it can be used as a user interface, while data gets stored at a remote location.

The networking solution between devices are also varied, but they mostly use the aforementioned wired and wireless protocols. Most households have a combined network device that has multiple functionalities, which in some cases are set up as separate dedicated devices. An IP router in a home network is a device that allows communication to the Internet via the Internet Service Provider's (ISP's) modem. An Ethernet switch is a network device that allows wired Ethernet devices to be connected together to a common medium while also making sure non-broadcast frames are only sent to the destination.

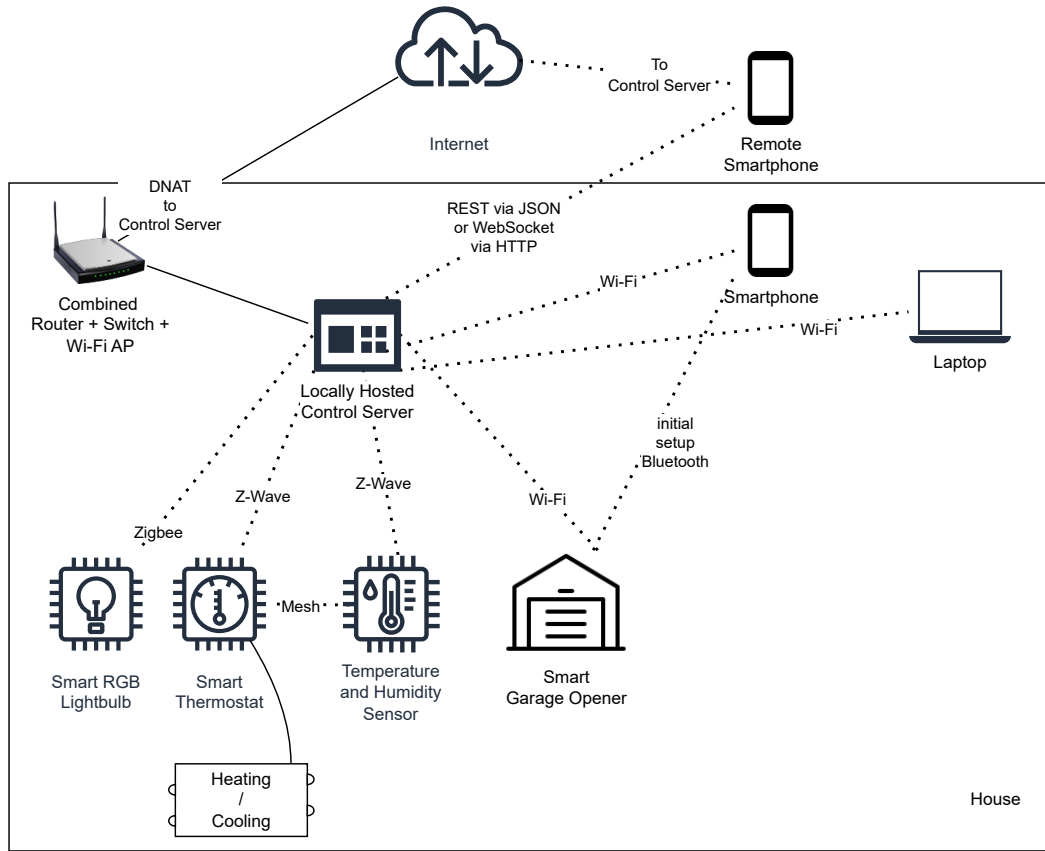


Figure 1.4: Architecture of a typical locally hosted smart home system

from outside the reach of the home via a firewall rule and NAT, or the home network can be reached via a VPN solution.

Besides containing electric parts, as their conventional equivalents, IoT-based smart utility devices utilize electronic parts, that make them "smart" by adding additional features. Special integrated circuits are added for basic computations, network connectivity, flash memory for firmware storage, additional circuitry for precise sensing and control of the higher power parts (eg. in case of light bulbs, garage openers, etc.). The latter function is most often implemented with usage of internal or external transistors and relays.

The surface, where users are able to control their smart home system besides occasional smart physical switches (eg. smart light switcher, smart thermostat) is the smart platform's user interface (UI). A well-designed and implemented UI is essential for optimal user experience and most solutions try to provide this. In most situations, the control server acts as a backend, which can be controlled via its REST API, or via a WebSocket connection for live data transmission. In both cases, the data is most often transmitted in the form of JSON messages, or for live video, WebRTC might be used. Frontends, or client-side applications can run on the users' devices, most often on smartphones, tablets, laptops and desktop computers via a web or native mobile platform application (eg. Android or iOS application). In case of mobile applications, they might also be able to be controlled via the device's set up voice assistant software using voice commands.

Chapter 2

Planning

The smart home model box's development and planning began during Project Laboratory and has continued since then with some variations, including the replacement of software platform. This chapter takes a look at the planning work, before the plans were put into practice.

2.1 General goals, requirements and principals of the planned model

The box project's goal was to get familiar with the most important IoT and related technologies, such as microcontrollers, networking, electronics control and to put this knowledge into practice via the creation of a smart home model system demo box, which can then be used to gain experience with the development and usage of such a smart home or home automation system.

During the box model project's planning and implementation phases, gaining a deeper insight of the underlying technologies and building a smart house solution from inexpensive, easy-to-obtain generic electronic components, microcontrollers and open-source software were favored over expensive, locked down out-of-the box solutions, as they are tuned for a quick and optimal end-user experience, but hide the implementation complexity from their users, therefore are less suitable for the project. The utilized software platform should be self-hosted in order to ensure absolute ownership of data and mitigate the vulnerability of being exposed to an external provider.

When finished, the box model should have electric and electronic devices inside it, which resemble utilities, that can be controlled along with sensors, detectors that collect the momentary status of different natural aspects at a given interval. It should also have a user interface, which can be used to control the appliances, display the current and historic readings of sensors accessible from a web browser or smartphone.

2.2 Hardware environment

The box containing the project's internals was chosen to be a shoebox, which I already had at hand and is large enough to have separated rooms and smaller electric and electronic components inside it, but is also small enough to be easily transported. The floor plan of the house wasn't fixed at the planning phase due to the uncertainty of the

picked devices' sizes, it was structured later in the implementation phase with cardboard to have separate rooms.

As for electronics inside the box, I decided on a single microcontroller board setup, where it is used to either control or read input from the connected devices and communicate to a server, which maintains all devices' status and provides a user interface. A microcontroller board is essentially a small computer, which is designed in a way to manage specific tasks within an embedded system without requiring a complex operating system. [29] Microcontrollers are well-suited for applications requiring real-time signal processing, such as controlling motors, servos and interfacing with various types of sensors and communications, therefore a great platform for home automation applications. The major microcontroller platforms considered for the project were Arduino, ESP and STM32. During Training Project Laboratory, I had the opportunity to use a NodeMCU ESP32S microcontroller, which is based on the ESP32S chip, that besides having a typical microcontroller functionality (ADC and DAC, PWM, UART and many more via their pins), also has wireless communication capabilities via Bluetooth and Wi-Fi. Due to this extra functionality often not being integrated in other microcontrollers, the cheap price and better price-to-value ratio, as well as the previous experience and easy access, I chose an ESP32-based NodeMCU ESP32S microcontroller board for the project. [33] [37]

One specialty of the selected NodeMCU microcontroller is its form factor, it has downward facing male pins, that can be plugged into a standard breadboard for easy connection to other electronics via jumper cables. Therefore many jumper cables and two breadboards were also obtained for the project, one breadboard for the microcontroller and one for higher power device circuitry featuring transistors, because the microcontroller has a specified current limit on its pins and wouldn't be able to provide enough power for such components.

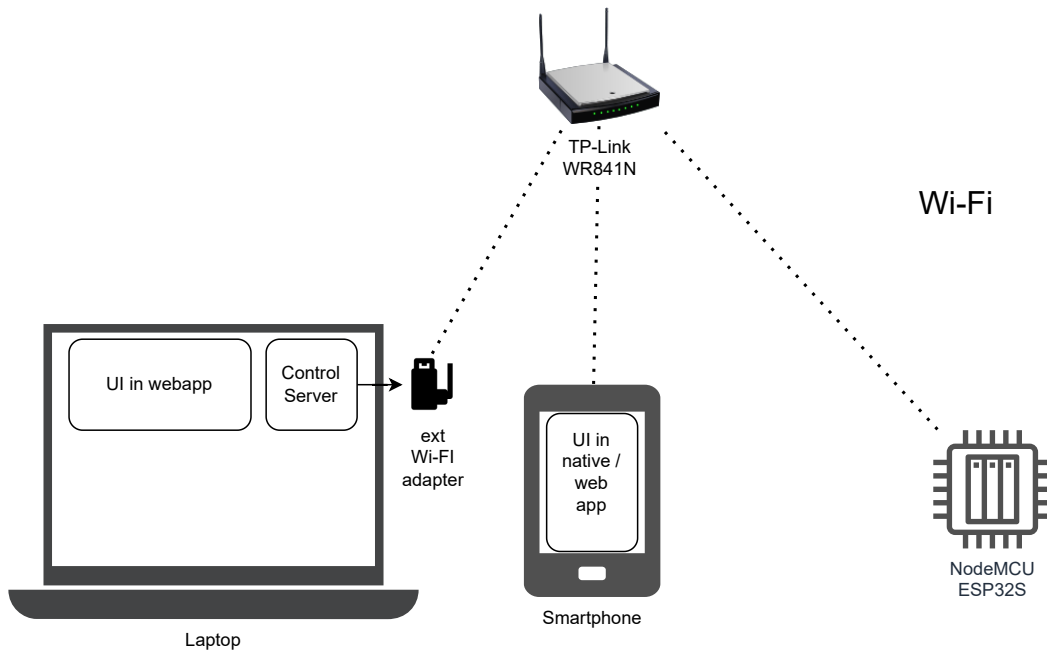


Figure 2.1: Planned model network architecture

The communication medium between devices was chosen to be Wi-Fi, as this is one of the wireless protocols supported by microcontroller and by most laptops and smartphones,

therefore a good common ground across devices. The control server was planned to be run on my own personal laptop for convenience and ease of development. Throughout Project Laboratory, I used a TP-Link WR841N at hand, which is a combined network device with routing, switching and Wi-Fi AP capabilities to symbolize a home network and provide a Wi-Fi network medium between devices. This remained in the the Thesis Project, with a custom firmware (DD-WRT) installed on it to have an open-source, more up-to date firmware, along with an extra Wi-Fi adapter plugged into the laptop for simultaneous usage of its internal Wi-Fi adapter to have Internet access. On figure 2.1, a diagram of this network setup is shown.

One of the most easily obtainable and connectable, yet spectacular electronic component that can be used in such a project is a Light Emitting Diode, or LED in short. A few LEDs and required resistors were acquired for the project in order to illuminate the rooms in a spectacular way, switched on or off from the user interface. Resistors of different common values were obtained to limit the current passing through components to protect them from overheating, for other devices as well, such as a light sensor in the form of an LDR (Light Dependent Resistor) and three LM235Z temperature sensors. For heating and cooling simulation, two 5V fans and a TEC1-12703 Peltier module were acquired. Two lower current rated BS170 transistors, two diodes and one higher current rated BD241C transistor were obtained for powering the two fans and the Peltier module from a higher-current (2A) external 5V power supply (a USB phone charger). And finally, an SG90 servo was acquired and fitted with extra parts to simulate an automated electronic rolling shutter.

2.3 Software environment

At the end of Project Laboratory, the software architecture of the model box was comprised of three main components: a JavaScript backend responsible for maintaining the components' status and communication between the microcontroller and user, microcontroller code for controlling the electronic components (reading sensor data, setting output for actuators), receiving and sending data to the backend and finally, a React-based frontend to send commands to and receive readings from the backend. For the Thesis Project, I decided on replacing it with already available open-source smart home system platforms with more functionalities and better device and community support, because I believe it would have been harder to further develop the earlier platform in an extensible way, add more features and the integration itself was also a great technical challenge to be featured as a Thesis Project.

Choosing the right smart home software platform for the project is an essential task, because it is the foundation of the project and other subsystems profoundly depend on it. Its features and offers, hardware and software support, community reception should be considered before fully committing to it, because changing to a different one can be tedious work if it doesn't meet the set requirements. After researching the major open-source smart home software platforms on the Internet, two projects were considered: Home Assistant and OpenHAB. [10] [34] Both projects have mostly the same to offer for a smart home platform: both are open-source, have good documentation, have built-in UI with customizations and integrations for many kinds of devices and options for automations. However, Home Assistant has a larger community of users, contributors and its popularity is also bigger (a Google Trends comparison also confirms this). [32] With these findings and recommendation from acquaintances in mind, Home Assistant was chosen to be the base smart home platform.

The microcontroller software platform is also important for the success of the project, it has to be compatible with the smart home platform, be reliable and has to have support for required device types and automations. Platforms compatible with Home Assistant for the ESP32S architecture were searched for and two major projects were found: ESPHome and Tasmota. Both focus primarily on supporting ESP-based microcontrollers, have integration for Home Assistant, mostly have the same popularity and community support and have support for Over-the-Air (OTA) updates. [21] [44] [22] [46] The main difference between the two projects are the technologies used for transmitting messages between the control server and microcontroller: ESPHome mainly uses Event Source API, but also has support for a simpler REST API and Tasmota uses MQTT, a message queueing protocol. [24] [45] Both use JSON payloads for their messages, however the technologies used by ESPHome usually make the propagation of control and other messages faster than Tasmota. Eventually, the microcontroller software platform was chosen to be ESPHome.

Chapter 3

Implementation

The implementation of the box model, according to the plans discussed in the previous chapter, mostly went without issues and was able to be put into practice. This chapter presents the steps of implementation taken in detail.

3.1 Floor plan

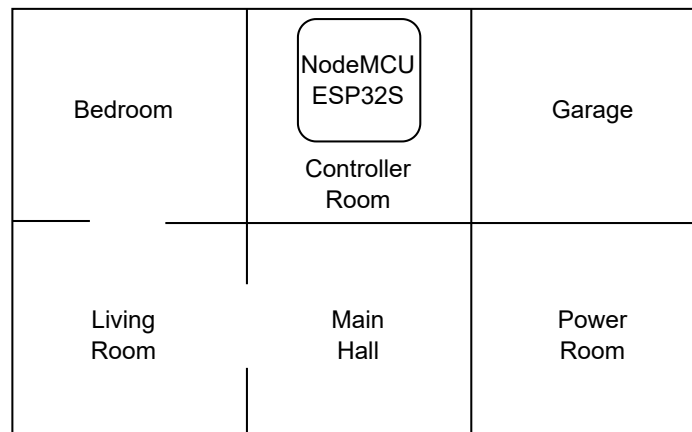


Figure 3.1: Floor plan of the finished box model

The shoebox used as a base for the project was structured with cardboard and scotch tape to have six equal-sized rooms inside it, shown on figure 3.1. Internal and external cutouts were made in the box to be able to push cables through them, have ventilation holes for the fans, and have a window for the rolling shutter visible from outside.

3.2 Connecting devices to the microcontroller

The NodeMCU ESP32S has many ports with different functionalities, as shown on figure 3.2. [1] It has a Micro-USB port that can be used for supplying power and connection to a computer for programming, flashing firmware and serial console. The 5V supplied from the USB port is connected to the VIN 5V pin and there is also a VIN 3.3V port from

PIN DEFINITION

www.ai-thinker.com

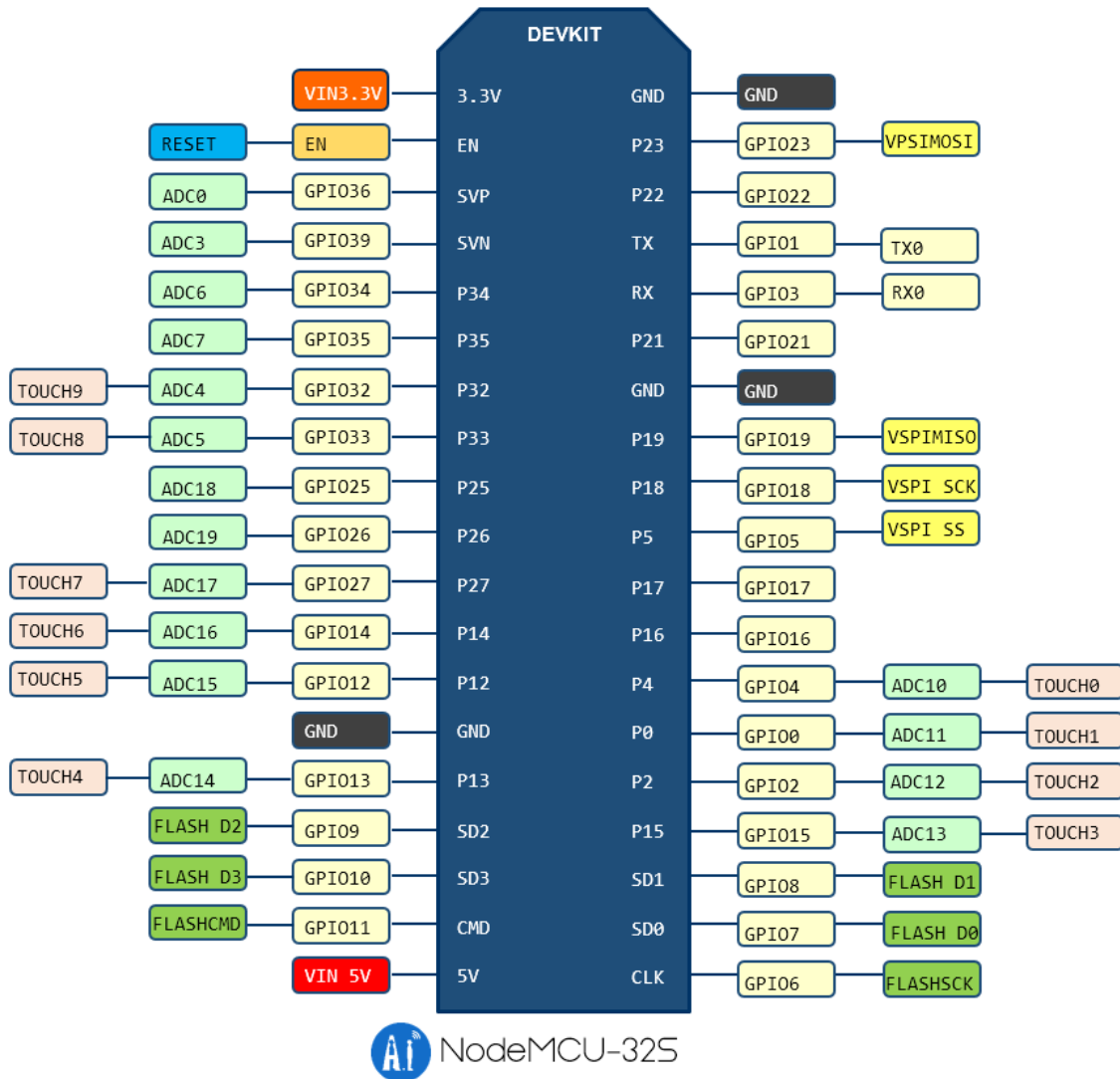


Figure 3.2: NodeMCU ESP32S Pinout

a voltage converter for supplying power to lower-voltage external components, besides the three ground (GND) pins. There are many more GPIO (General-Purpose Input/Output) pins with different input-output functionalities, however not every of them can be used for every purpose. For example, ADC/DAC pins can be used for Analog-Digital Conversion or vice-versa, therefore sense or create voltages between 0V and 3.3V, some can output PWM (Pulse Width Modulation) signals for dimming LEDs, controlling servos etc., serial pins can be used for serial communication, SD card interfacing and there are some only binary input-output pins. The 3.3V, 5V and ground pins were connected via short bendable solid cables to the power rail sides of the main breadboard for easy distribution of power to components. Jumper cables and wires were used to connect most components to the breadboards, which also served as extension to have appropriate length of cables for them and to be able to put into their desired location inside the box. The method for cable splicing varied, for some, soldering and shrink tubes were used and for others, only wire

twisting and electrical tape was used due to the smoke detection system installed in the dormitory, where the project was mostly made.

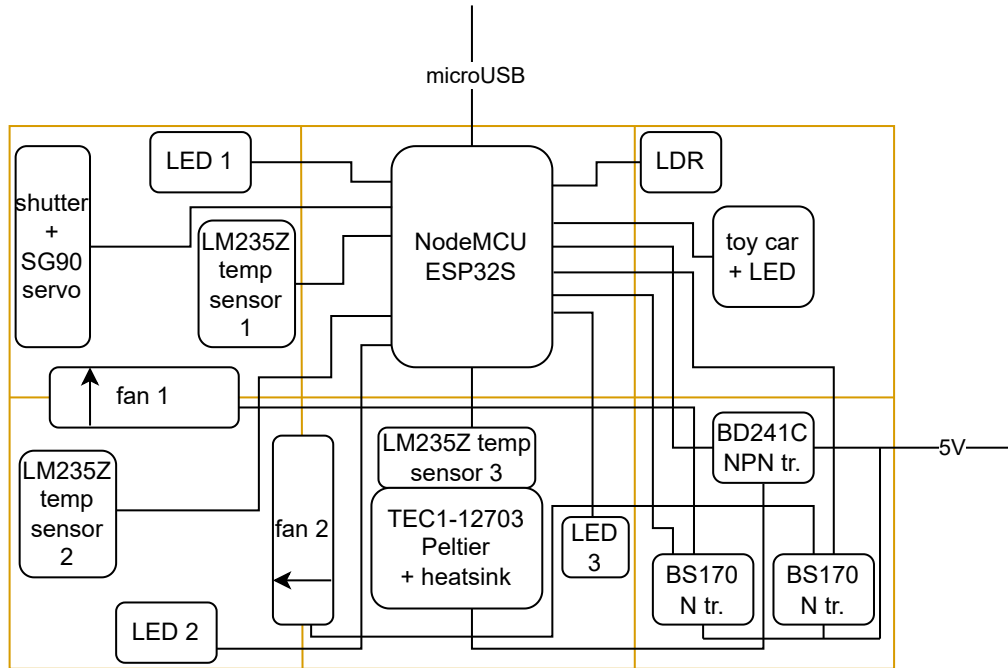


Figure 3.3: Simplified block diagram of components and their wirings inside the box

The microcontroller's number of pins and their functionalities were adequate the project, extra components can be added up to the limit the available free pins and the required features on them. The pins used for room LEDs were chosen to be non-ADC GPIO pins and for each one, 100Ω resistors were put between the LED's anode and the microcontroller pin, the cathode was connected to the side ground rail of the breadboard. An LED put inside a small toy car was also connected to a PWM-capable pin with 50Ω resistance in series to indicate a simulated electric car charging, based on the readings of the light dependent resistor (LDR). The LDR's wiring required a different approach: one end connected to the 3.3V rail, and the other to one of the microcontroller's ADC pins, and to the ground with a $10k\Omega$ resistor. The three LM235Z temperature sensors were connected with yet another wiring method: $2k\Omega$ resistors were put between 5V and their V+ terminals, to this the ADC pins were also connected and finally, the V- pins were connected to ground. The SG90 servo was connected to 5V, ground and a PWM capable pin for control. Due to the limited space remaining on the first breadboard, a second one was utilized for the fan and Peltier powering circuit. The ground between the two breadboards was connected with a jumper cable, and a USB cable with male jumper plugs spliced was created to utilize external power from a 5V 2A phone charger. The 5V fans are powered and speed controlled by BS170 transistors utilizing pulse width modulation (PWM). PWM-capable pins of the microcontroller were connected to the transistor's gate terminal, along with $10k\Omega$ resistors to ground, the source terminal connected to ground and diodes placed between 5V and drain to protect the microcontroller from the inductive load of fans by only allowing flow of electricity in one way. The fans' negative terminals were connected to the transistors' drain and the positive to 5V. Finally, the TEC1-12703 Peltier module has a small heatsink taped onto it and is powered and controlled by a

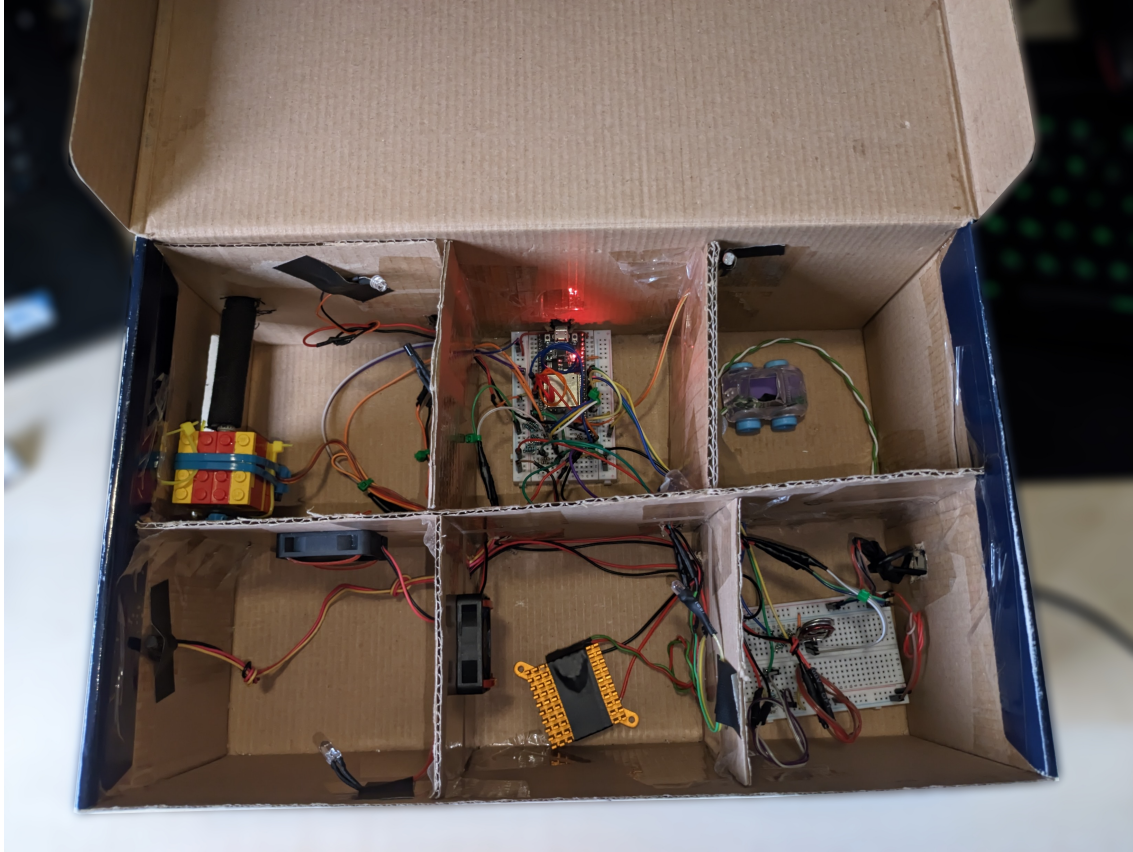


Figure 3.4: Photo of the finished box model

BD241C transistor (with extra cooling attached in the form of soda can tab openers screwed into it for more surface area). This Peltier module's maximum operating voltage is rated to be 15.4V, which means, when run at a lower voltage, it draws less current, therefore functions with a lower power. According to its datasheet, at 5V (which is given by the USB charger), its current is approximately 0.8A, therefore the device draws about 4 watts of power and it decreases with more ambient temperature as the current gets less. [28] 333Ω resistance was used between one of the DAC pins and the transistor's base, the emitter was connected to ground and the Peltier's terminals were connected to the collector and 5V. In figure 3.3, a simplified block diagram of components and their wirings inside the box is shown without resistors, ground connections, exact pinout of components, and on figure 3.4, a photo of the finished box model is shown with the microcontroller turned on.

3.3 Network setup

The Wi-Fi communication medium was set up using a TP-Link WR841N combined network device, targeted for small or home office use. Such devices almost always feature a web-based user interface for their setup and configuration. This was used to flash DD-WRT on it, which is an open-source Linux-based router firmware targeting a variety of Wi-Fi routers and embedded systems. [17] It was then set up to mimic a typical home environment, with an SSID of `esp_smart_home`, WPA2-PSK authentication, without internet connectivity and without NAT (Network Address Translation, by setting the operation mode from gateway to router). The IP address range remained the default

192.168.1.0/24 and the router's IP also remained the initial 192.168.1.1, due to it not colliding with other networks actively used on the laptop and smartphone.

An extra Realtek chip based USB Wi-Fi adapter at hand was used for connecting the laptop to the Wi-Fi network, with a custom network profile configuration. Initially, the use of this adapter was done with a network profile limited to the specific network adapter and manual IP configuration was used instead of DHCP to ensure a fixed IP address of 192.168.1.100 and no default gateway learned from this network. Later, it was passed through as a USB device to a virtual machine (VM) with the same Wi-Fi and IP configuration. The Android-based smartphone used for the project was also connected to this network, but didn't have internet connection despite having an Internet-capable SIM card put into it meant as a fallback connection, according to the gained experience.

3.4 Software setup

The chosen software environment discussed in the planning phase was suitable for the usage of obtained hardware, this section showcases the steps taken for its software setup. The personal laptop used for development had a Linux distribution installed, the smartphone used for testing ran Android 14.

3.4.1 Home Assistant initial setup, migration

Initially, Home Assistant was set up in a Docker container environment running on the laptop, with a Docker Compose file specifying the parameters for the container. Docker Engine is a container runtime engine, that makes it possible to run prepackaged container images utilizing the host machine's kernel with minimal overhead and adequate isolation. [18] The container images contain a Linux operating system's system libraries, executables and the application itself, therefore provide an easily reproducible, uniform runtime environment despite the differences in various systems. By itself, Docker has a command line interface to run and manage its containers, which means it has to be parameterized from the command line, or put into a script in its parameterized form and executed. Docker Compose is a solution for defining and running multiple Docker containers using an easy-to-manage definition, put into a YAML file. [19] YAML is a human-friendly data serialization language with a really simple syntax, as shown in listing 3.1. [50]

```
shopping_list:
  - bread
  - milk
  - chocolate

# ideal room temperature
temperature: 22

winter_months: ["December", "January", "February"]
```

Listing 3.1: YAML file example

In listing 3.2, the Docker Compose definition of the set up Home Assistant container can be seen: 2024.10.3 is the selected version (tag) of the image is used, the config directory from the compose file's directory is mounted directly into the container's file system as /config along with the host machine's local time and dbus for time and device access, and finally, socket connections are bound on the host machine's 8123 port on the Wi-Fi card and localhost interfaces to access the same port inside the container.

```

services:
  homeassistant:
    container_name: homeassistant
    image: "ghcr.io/home-assistant/home-assistant:2024.10.3"
    volumes:
      - ./config:/config
      - /etc/localtime:/etc/localtime:ro
      - /run/dbus:/run/dbus:ro
    ports:
      - 192.168.1.100:8123:8123
      - 127.0.0.1:8123:8123

```

Listing 3.2: Docker Compose file used for the initial Home Assistant environment

Even though only one container is defined in the compose file, it still shows Compose's advantage, as the environment can be run with a simple command in the terminal, when inside the compose file's directory:

```
docker compose up
```

This setup in the Docker environment functioned properly, however had one limitation: Home Assistant add-ons aren't supported in the container variant, only in the Home Assistant Operating System (HA OS) or the Supervised installation (installed on top of a Linux installation) variants. [12] A voice assistant was a sought after feature in the project, therefore it was migrated to the HA OS variant, run inside a virtual machine, or VM.

Virtual machine hypervisors allow physical, or host computer resources to be shared to multiple virtual machines, or guest machines. [49] Each virtual machine can run its own separately with its own amount of CPU cores, RAM, storage and other kinds of resources. Virtual Machines are widely used today in IT infrastructures, as the technology makes it easier to create heterogeneous software environments and can offer less downtime with clustered servers. The most popular current virtualization solutions include VMware products, Oracle VirtualBox and Proxmox. [47]

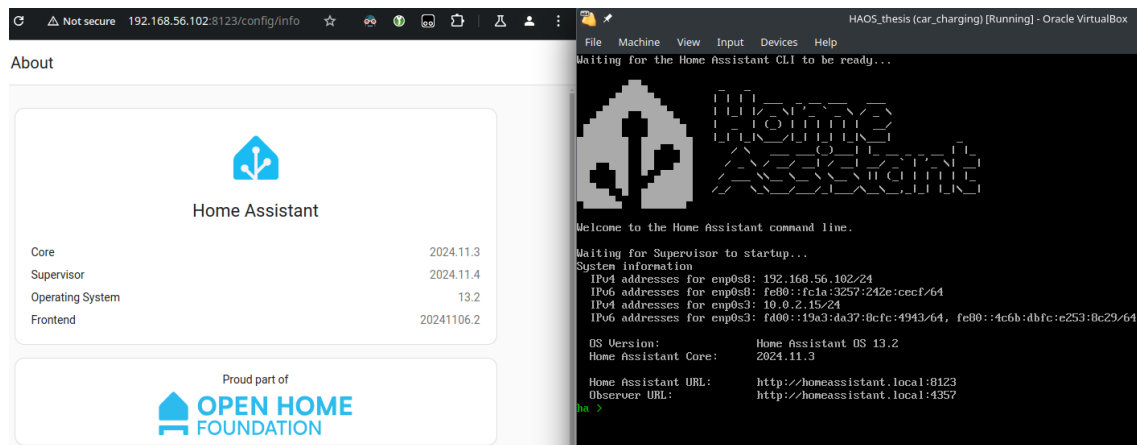


Figure 3.5: Home Assistant OS running in VirtualBox with its web GUI opened from a browser

To run HA OS, the selected VM hypervisor solution was Oracle VirtualBox, due to it being free and the previous experience gained with it in the past. HA OS is distributed in different disk file formats for various hypervisors, VDI was used for VirtualBox. Initially, 2 virtual CPU cores and 2 GB of RAM of the host's 16 GB was assigned to it, however it was later changed to 4 GB of RAM to run the voice assistant functionality faster. Two virtual

network adapters were added to the VM: one in NAT mode, to provide Internet connection for updates and downloading add-ons, and one adapter in Host-only adapter mode, to be able to be reached from the host (on it's web UI (8123) and SSH (22) ports). Most VM hypervisors offer USB passthrough to virtual machines, which hides the USB device from the host and makes it seem as it was connected to the VM directly. USB passthrough was used to access the USB Wi-Fi adapter inside the VM and the network credentials were set via a parameterized `ha network` command with the specific SSID, password and IP configuration. The SSH add-on was also installed to access command line configurations from a client terminal installed on the host operating system. The configuration of the new environment (eg. integration and dashboard setups) didn't need to be redone from scratch, as the old environment could be exported to a tar backup file, which was then able to be imported to the new HA OS environment without issues during its initial setup from the web GUI.

3.4.2 ESPHome

To get started with installing ESPHome on the microcontroller, its official getting started with command line guide was followed. [20] A sample configuration YAML file was created using the wizard command. The ESPHome compiler and flasher was run from a Docker environment, first flashed via USB and later updated wirelessly using ESPHome's the Over-the-Air (OTA) functionality via Wi-Fi.

```
# for first flashing, add: --device=/dev/ttyUSB0
docker run --rm -v "/esphome":/config -it ghcr.io/esphome/esphome:2024.11.1 run smarthome.yaml
```

Listing 3.3: Docker command to compile and flash ESPHome via OTA / USB

By default, an ESPHome configuration file only consists of one main YAML file, which contains all configurations. On listing 3.4, an excerpt of a configuration file is shown generated by the wizard for the NodeMCU ESP32S microcontroller.

```
esphome:
  name: livingroom
esp32:
  board: nodemcu-32s
  framework:
    type: arduino
ota:
  - platform: esphome
    password: ""
wifi:
  ssid: "homenetwork"
  password: "teriyaki-noodles"
```

Listing 3.4: Excerpt of a configuration file generated by the ESPHome wizard

After successfully flashing the compiled firmware to the microcontroller, it can then be added to the Home Assistant environment with an "integration". Integration is a term in Home Assistant's ecosystem to connect and integrate other software and hardware platforms, in our case ESPHome to the Home Assistant environment as a device. [7]

```
switch:
  - platform: gpio
    id: bedroom_led
    pin:
      number: GPIO21
      mode: OUTPUT
    name: "BedroomLED"
    restore_mode: ALWAYS_OFF
```

Listing 3.5: ESPHome configuration for a simple LED switch

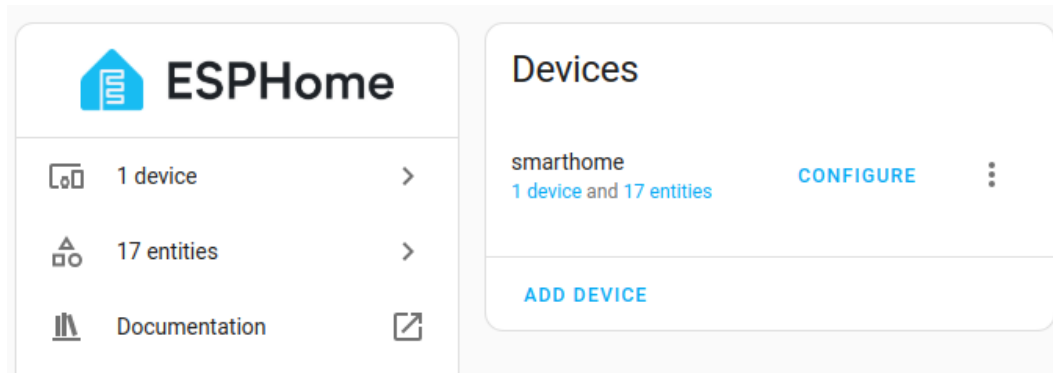


Figure 3.6: ESPHome integration for Home Assistant set up

ESPHome supports many types of electronic components connected to the microcontroller’s pins (as the project’s homepage shows), with a relatively easy configuration. [21] To set devices up to be utilized by ESPHome, their configuration has to be defined in the YAML configuration file, as shown in listing 3.5 for a switch, that toggles an LED. Each device should have an id and name set, along with platform specific details, such as the GPIO pin number and mode (input / output).

This configuration was also done iteratively to the other connected components as well, but in separate files with the use of packages, which is a functionality provided by ESPHome for better code structuring and maintainability. [23] With packages, ESPHome configurations are merged from multiple YAML files to a single one during compilation, as if they were written in that single one.

```
packages:
  fans: !include packages/fans.yaml
  ldr: !include packages/ldr.yaml
  leds: !include packages/leds.yaml
  peltier: !include packages/peltier.yaml
  shutter: !include packages/shutter.yaml
  tempsensors: !include packages/tempsensors.yaml
```

Listing 3.6: ESPHome package includes in the main configuration file

After reconfiguring the microcontroller with extra devices, Home Assistant automatically takes care of the devices added and new entities are created for them. In Home Assistant’s terminology, entities are the basic building blocks that hold data and are used to monitor physical properties or control entities. [7] On figure 3.7, a few entities from the ESPHome integration device are shown.

The ESPHome integration’s device, its controls, sensors and configurations can also be seen and configured, as shown on figure 3.8.

One component connected to the microcontroller was harder to be configured for use, than others, which is the SG90 servo used for the rolling shutter. Fortunately, other ESPHome users had also tried to make this exact servo model work and one forum member was able to and provided configuration code for it, which includes PWM configuration, conversion of value from a number slider’s -100 to 100 range to the servos 180 degrees of control and a sensor for the currently set position. [25] This example superbly shows an advantage of open-source software and its communities, how projects can be easily extended with extra, more complex components and how users can get support for their problems.

	↑ Name	Entity ID	Integration	Area
🌡️	Bedroom Temperature	sensor.bedroom_te...	ESPHome	Smart Home
📱	BedroomLED	switch.bedroomled	ESPHome	Smart Home
⚙️	Brightness Sensor	sensor.brightness_s...	ESPHome	Smart Home
💡	Car Charging LED	light.carchargingled...	ESPHome	Smart Home
🌀	Fan 1	fan.fan_1	ESPHome	Smart Home
🌀	Fan 2	fan.fan_2	ESPHome	Smart Home

Figure 3.7: Home Assistant entities provided by the ESPHome integration

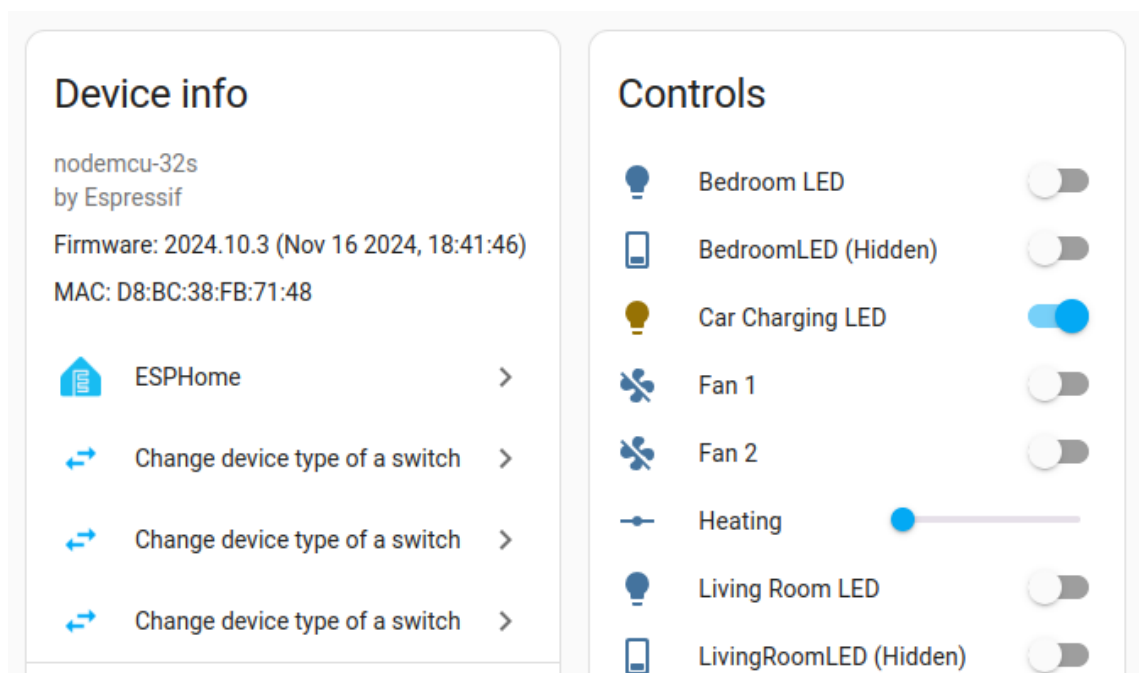


Figure 3.8: Device of the ESPHome integration and its control entities

3.4.3 Additional user interface setup, customization

3.4.3.1 Entity customization

Customizations can be applied on Home Assistant entities to better suit the user's needs. Each entity has a unique id, which is used as a basis to uniquely identify a specific entity and it is guaranteed to be static and never changes. [9] When present, it gets assigned an entity id, which is in the form of <domain>.<id> and is used as a logical identifier

< Fan 1

Name

Icon

Entity ID*
fan.fan_1

Add label

Voice assistants

Configure aliases and expose settings for voice assistants

Enabled

Disabled entities will not be added to Home Assistant.

Visible

Hidden entities will not be shown on your dashboard or included when indirectly referenced (ie via an area or device). Their history is still tracked and you can still interact with them with actions.

Use device area (Smart Home)

You can [change the device area](#) in the device settings

DELETE

UPDATE

Figure 3.9: Entity customization in Home Assistant

in dashboards, automations and others. [8] To give examples, `fan.fan_1` is in the `fan` domain, therefore can be controlled as a fan, has an id of `fan_1` and `light.bedroom_led` is a light with an id of `bedroom_led`. The entity id can be changed if desired, but it has to be unique throughout the Home Assistant environment. And finally, each entity has a name, which is used as a display name and is displayed on dashboards, can be referred to in the voice assistant, etc. Besides a name and id, an icon and labels can also be specified, as shown on figure 3.9.

Helpers are software tools to add or change functionality of different entities. For example, the states of multiple devices can be combined to a single value (eg. to calculate the average temperature in the house), change the type of a switch to a different device (eg. toggleable binary light, fan, lock) or combine multiple entities into a single virtual entity with the usage of the group feature. On figure 3.10, the helpers used in the project are shown.

3.4.3.2 Dashboard customization

Multiple dashboards can be created in Home Assistant to control and show different aspects and devices of a house. Upon setup, a default dashboard is automatically created, the layout of which is shown on figure 3.11.

On the default dashboard, all areas, device controls and readings, integrations are shown as a whole, which can be ideal for a typical home environment with multiple devices, but not for the project with basically a single device and its many entities. Therefore another dashboard with a sections type view was created, which is shown on figure 3.12. Multiple






	Average temp sensor.average_temp · Combine the state of several sensors
	Bedroom LED light.bedroomled · Change device type of a switch
	Living Room LED light.livingroomled · Change device type of a switch
	Main Hall LED light.mainhallled · Change device type of a switch
	Room Lights light.room_lights · Group

Figure 3.10: Home Assistant helpers used in the project

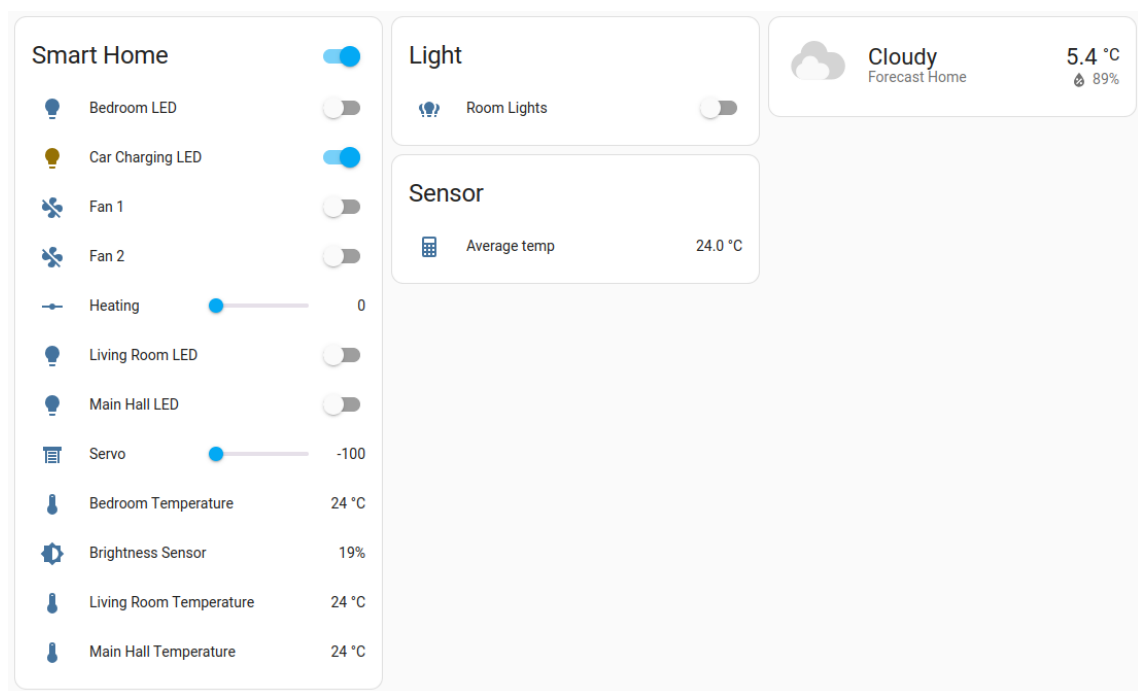


Figure 3.11: Default dashboard layout with controls and readings for the box device and weather forecast integration set to Budapest

sections were created for the various types of controls (eg. lights, fans) and sensors (room temperatures), which were grouped together by the types of entities. Additional extra entities were added three sections for display: the average temperature of the three rooms to the "Temperatures" section and the LDR sensor's value to the "Room Lights" and "Car Charging" sections. The car charging section mimics electric car charging with imaginary

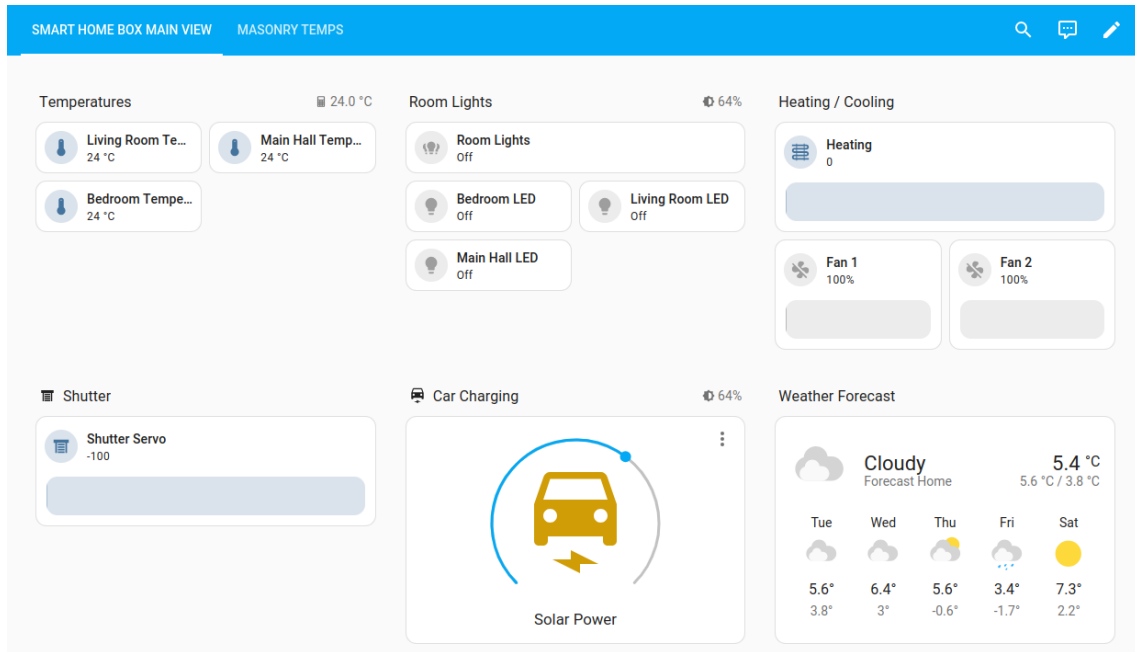


Figure 3.12: Custom sections dashboard with devices grouped together by type and weather forecast integration set to Budapest

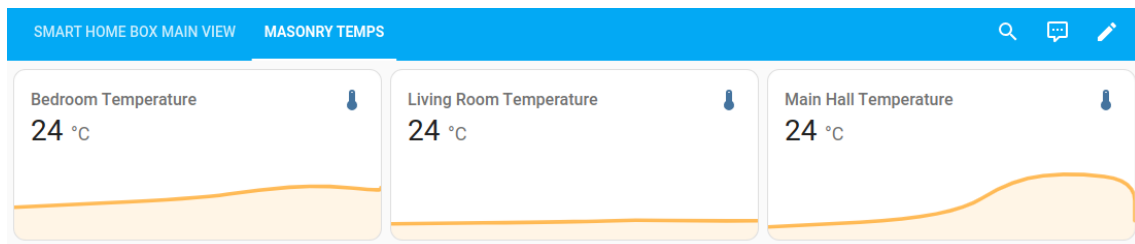


Figure 3.13: Masonry dashboard layout with historical graphs for room temperatures

solar panels (actually based on the value of the LDR): the set brightness of the automated LED is used as the value for display, click events on the section were disabled to make it read-only (from this dashboard).

A masonry type view was also added to the dashboard, which is shown on figure 3.13. When multiple views are present for a dashboard, a tab strip is shown to select between them, as shown on the screenshot. On the masonry view, added cards are sorted in columns based on their sizes, three equal sized historical graphs were added of the room temperatures recorded.

3.4.4 Smartphone setup

Besides having a web-based UI, Home Assistant also has mobile applications for the Android and iOS platforms called Home Assistant Companion. [6] To test the mobile application, it was installed to a personal Android smartphone. The smartphone was connected to the Wi-Fi network with its credentials, and the IP address of the HA OS VM Wi-Fi adapter with the proper protocol and port (<http://192.168.1.100:8123>)

was entered into the application as the instance URL. On figure 3.14, a screenshot of the Android application is shown.

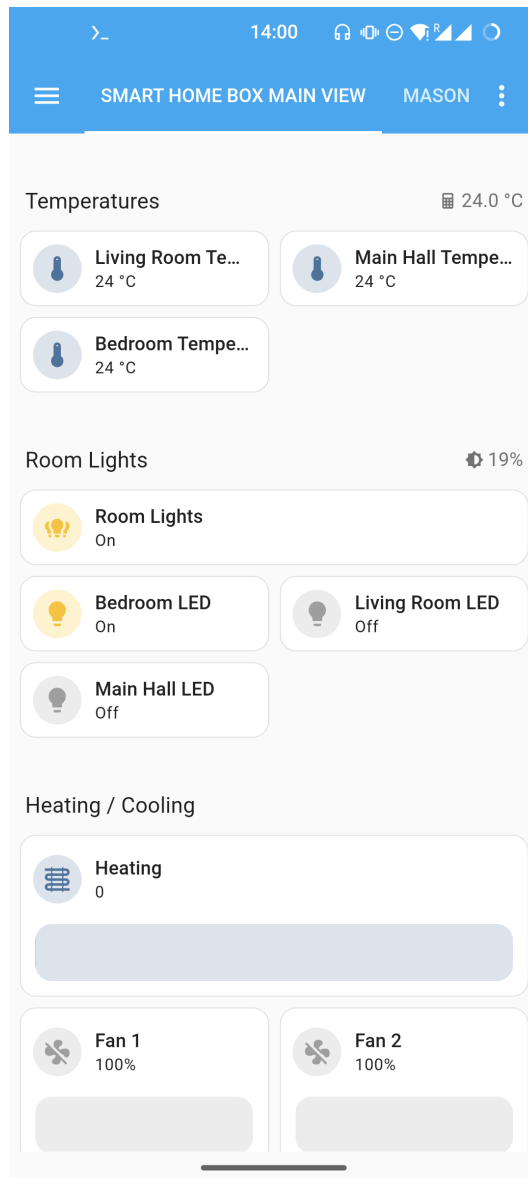


Figure 3.14: Home Assistant Companion app running on Android
14

Contrary to expectations, simultaneous use of the the Internet-less Wi-Fi network and the phone's data-enabled SIM card for Internet connection didn't work, even if the "Mobile data always active" option was turned on in the developer settings. A message is shown when connecting to the network about it not having an Internet connection and whether the phone should stay connected to it and only upon pressing yes, can the server be reached. But at the same time, the status bar shows that LTE disappears, which can mean that the radio data connection is dropped or only the routing table is changed, but in such a way that the default gateway gets overridden to the Wi-Fi network. It would likely be possible to be able to change this behavior with a rooted phone (where a root shell access is present), but its risks and disadvantages aren't worth it for this simple test scenario. And in a typical home network environment, where there wouldn't be a need to

access two networks simultaneously, the HA OS server and the Internet would be able to be reached at the same time.

3.4.5 Automations

To make the life of its users easier and more comfortable, Home Assistant offers automations to run specific tasks. An automation comprises of three parts: triggers (when it runs), conditions (optional, only run if these conditions are met) and actions (what actions to perform). [5] Besides basic automations, there are further features closely related to them: scenes can be used to set a group of entities to a specified state, scripts can contain multiple actions and blueprints can be used to specify a generic automation framework, which can be used as a common basis for multiple different automations with exactly specified entities.

During Project Laboratory, one feature that the model box had is the ability to show "solar power" by changing the brightness of the LED put inside a toy car based on the readings of the LDR. This was possible to replicate in Home Assistant too, with the use of an automation. Two kinds of triggers were tried for detecting when the automation should run: the first was using "Brightness Sensor voltage change", however even with a very small duration specified (1-5 seconds) between 0 % and 100% set as the voltage percent range, the automation wasn't triggered frequently enough, only about once a minute. Therefore an other trigger was used: a periodic interval of 5 seconds, which works reliably and its configuration is shown on figure 3.15.

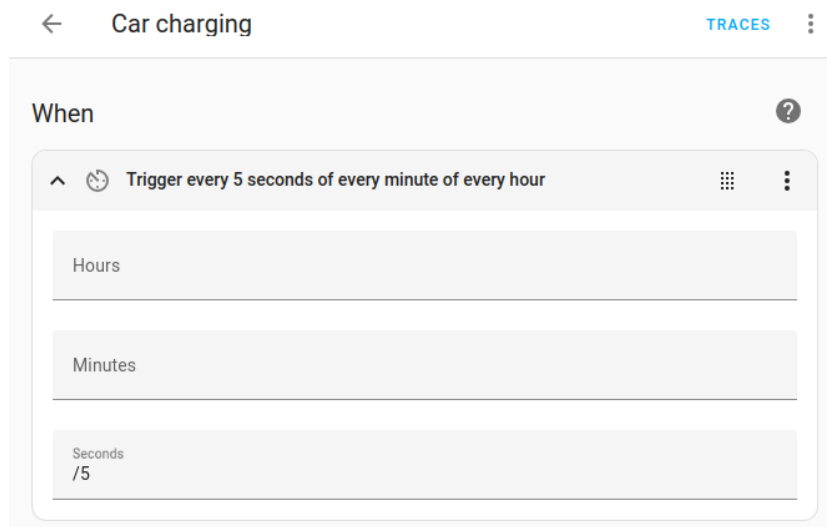


Figure 3.15: Home Assistant automation trigger

The action needed to accomplish one simple task: set the brightness value of the car charging LED, based on the readings of the LDR. This was possible to be defined via the YAML editor (the set content is shown in listing 3.7), as the visual editor would have only allowed a constant number to be set as brightness. For action, `light.turn_on` was specified, that turns on the specified target light entity (which is `light.car_charging_led`, if it isn't on already) and a `brightness_pct` data key is also specified, which tells `light.turn_on` what the target light's brightness should be set to. This key is defined to be the result of a template (inside double curly braces): `states()` retrieves the value of the LDR reading, which is in the form of an integer between 0 and 100 and it gets set to be `brightness_pct`.

```
action: light.turn_on
metadata: {}
data:
  brightness_pct: "{{ states('sensor.brightness_sensor') }}"
target:
  entity_id: light.car_charging_led
```

Listing 3.7: Home Assistant automation action for setting LED brightness based on LDR

3.4.6 Voice assistant

As a prerequisite for the locally hosted voice assistant, the Home Assistant Docker environment was migrated to HA OS running in a VM, explained with more details in chapter 3.4.1. To set up the voice assistant environment, the installation guide was followed to set up the Piper (text-to-speech) and Whisper (speech-to-text) add-ons. [11] The add-ons were configured to use a low quality speech detection and synthetization, in order to ensure fast response, as even with medium, the response time dramatically increased after pronouncing the commands. A screenshot of the assistant can be seen on figure 3.16. The speech detection and synthetization supports many languages to be configured: English (UK, US), German and Hungarian were tried and out of these, English had the best accuracy of understanding and executing the said commands. Although it is impressive, it only works with simple commands, when exact device names are specified and doesn't reach the level of Artificial Intelligence based chatbots, that gained a rapid technology boom in the last few years.

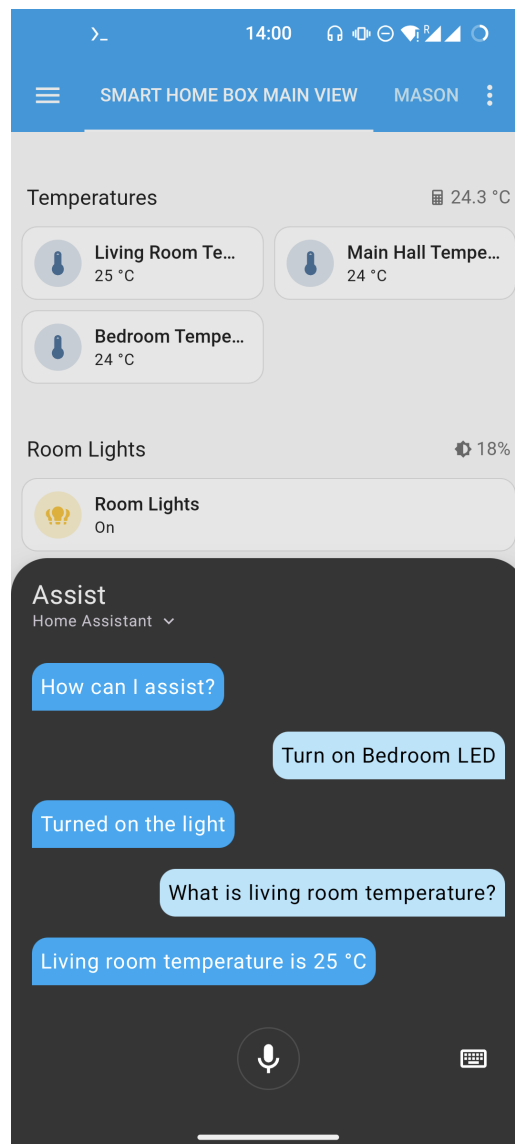


Figure 3.16: Screenshot of Home Assistant's locally hosted voice assistant in action

Chapter 4

Review

The last chapter mainly focuses on the box model's evaluation to the requirements specified during the planning phase, shortcomings and how these could potentially be overcome and finally, opportunities of further development and additional ideas.

4.1 Usage

During usage, controlling the model box's components via the user interface is quite impressive, as when components are toggled or values are altered from a laptop or smartphone, the changes made propagate really fast to the model, in a few seconds at most. Besides control, monitoring of the implemented sensors is also reliable, as when influenced externally eg. less or more heat, light are applied to them, a change can be seen in the readings of sensors. These values, along with other metrics, actions taken by the user are continuously logged and this history can be reviewed later, or even be plotted to a graph in case of quantifiable amounts, between certain time ranges.

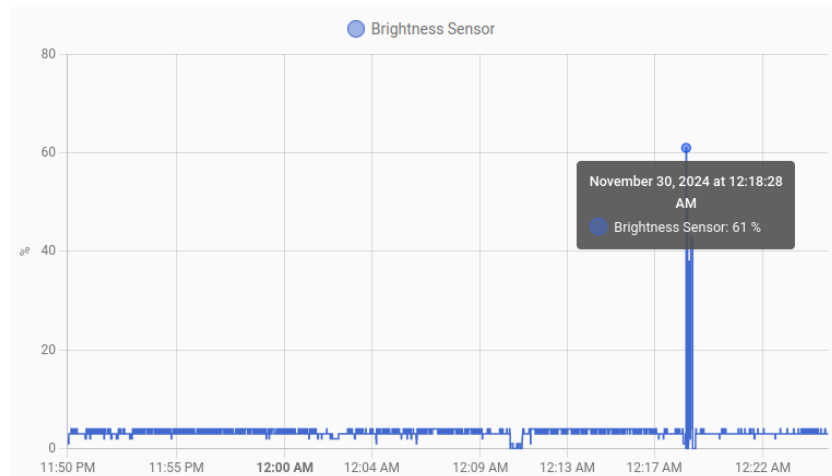


Figure 4.1: Home Assistant history graph for the LDR (in a dark environment, the outstanding values show when a flashlight was shined upon it)

The use cases of the model include the most important aspects of a typical smart home environment: monitoring of temperature and light levels, control of basic utilities in the

form of heating and cooling, and lastly, a "smartified" rolling shutter (that would be actuated by hand in a conventional home).

4.2 Critical analysis, shortcomings

The experience of using the model box and its user interface is really similar to what a deployed home environment would be, therefore an adequate way to simulate a real smart home system with certain limitations. The model features a full-fledged smart home software platform, which is used in many installations worldwide. The number of "devices" (according to Home Assistant's terminology) is less, than in a usual real environment with more heterogeneous devices, but the number components connected to it makes it comparable to that of a basic installation. There are aspects however, that are usually part of a typical installation, but weren't featured in the model, these are security management and energy monitoring systems (besides the solar power car charging simulation), due to the limited scope and resources of the project. They could be simulated with more components added, but it would have increased the project's cost, it is also possible that an other microcontroller would have been needed.

With the selected software platforms and devices, the user has absolute control over their smart home infrastructure and its data. With the proper knowledge, it can be changed, further developed and expanded with additional modules due to its open-source nature. In a typical installation, the smart home platform would be run on an always running computer, server or low-powered microcomputer (eg. Raspberry Pi), along with an uninterrupted power supply (UPS) to prevent data loss and quicker recovery from momentary power disruption, instead of running from a laptop, that always transported to different locations. However, the system's operation and management requires basic IT system administration knowledge, that many smart home end-users lack, therefore this solution might not be favorable for them and should utilize easier to set-up, out-of-the-box solutions, or hire a contractor, who is specialized in smart home system installations and maintenance.

A smart home platform, its network communication medium and connected devices, appliances should function reliably. If this is not the case and one or more utilities aren't available, their unavailability can cause different levels of inconvenience to the inhabitants of the house. This can range from minor sensor readings not available or updated, to lights or hot water for shower not available, to the lack of heating during winter or air conditioning during summer.

A smart home platform should also have adequate security measures, as its security is only as strong as its weakest link. The demo box uses a separated Wi-Fi network with a decent pre-shared-key (password) and encryption (WPA2), and the control server is only accessible from this Wi-Fi network or the development laptop and requires authentication with custom credentials set. It is not accessible from the Internet (it is behind Network Address Translation), but a typical environment would be set up for access, therefore security is even more important in that case. If such systems are hacked into, it can have varying negative consequences to their owners: starting from toggling and changing values of appliances, gatewaying into the home network via a compromised device to find additional vulnerabilities, to theft of data and spying in the form of video or audio means (if a camera or voice system is installed). Therefore it is crucial to ensure security, software updates should be installed regularly (and these shouldn't introduce new vulnerabilities or bugs, as it sometimes happens by some irresponsible manufacturers).

The model box’s hardware doesn’t suffer from vendor lock-in, as it often is with many commercial smart home solutions. The platform is modular, parts can be substituted, added and further developed thanks to its open-source nature. However, changing the software platforms can still be tedious and time-consuming, as the environments need to be set up again and customized to the specific needs of the users and exact environment, and can take between hours to days in a typical home installation.

4.3 Opportunities of further development

The model can be further developed in its current hardware form with additional software features, eg. with an automatic thermostat set to a target temperature and dashboard improvements. With additional hardware components, the currently missing subsystems (eg. security with card readers and RFID cards, energy monitoring with a power meter for electricity and its continuous logging) can be added, however this might require the utilization of a second microcontroller (which ideally should have the same microcontroller architecture), due to the limited number of remaining free pins and the features that these provide (eg. PWM, ADC, DAC) not being enough on the first.

An even larger leap for the project would be to be deployed in a real house in a more production-grade environment. The controller software should be installed on an always-on computer, server or microcomputer (eg. Raspberry Pi). It should be connected to an uninterrupted power supply (UPS) for less downtime during power outages. The devices utilized would be larger, higher power specialized appliances with a hardware platform capable for controlling electricals, communicating to the control server and firmware with integrations to the specific smart home platform, optionally changeable to an open-source variant.

Newer, more efficient IoT communication protocols and devices can be also tried out and added to the system, such as Z-Wave, Zigbee and Matter. They can achieve less power consumption, less crowding of the ISM bands (as many Wi-Fi, Bluetooth devices tend to overload the 2.4 GHz band), mesh networking to achieve better range and more compatibility between devices from different manufacturers.

Besides trying out new protocols and technologies utilizing them, there is also a need for further development of these software and hardware solutions in the field of IoT and home automation. The demand for such solutions is steadily increasing, therefore the development and operation of these solutions can be a profitable business idea, as noted in the introduction.

Acknowledgements

Firstly, I would like to thank and acknowledge the work of both of Home Assistant's and ESPHome's project participants, developers and collaborators. These projects have helped me a lot throughout the project. Even though I had used my own code in Project Laboratory with mostly the same physical hardware, it would have been harder and less reasonable to further develop and refine that in the thesis to have the same current features available, in a more easily extensible, modular manner manner, support for many types of devices and products.

Additionally, I would like to thank my advisor Róbert Schulcz for his work and support that he has been providing for me since Training Project Laboratory.

Bibliography

- [1] AI-Thinker. NodeMCU-32S Core Development Board Documentation. URL https://docs.ai-thinker.com/en/esp32/boards/nodemcu_32s. Accessed on: 2024-11-18.
- [2] Amazon. Alexa Smart Home. URL <https://www.amazon.com/alexa-smart-home/b?ie=UTF8&node=21442899011>. Accessed on: 2024-11-03.
- [3] Andrew Gebhart Angela Moscaritolo. What Is Matter? The Smart Home Standard, Explained. URL <https://www.pcmag.com/explainers/matter-explained>. Accessed on: 2024-11-02.
- [4] Apple. Developing apps and accessories for the home. URL <https://developer.apple.com/apple-home/>. Accessed on: 2024-11-03.
- [5] Home Assistant. Understanding automations. . URL <https://www.home-assistant.io/docs/automation/basics/>. Accessed on: 2024-11-27.
- [6] Home Assistant. Companion Docs, . URL <https://companion.home-assistant.io/>. Accessed on: 2024-11-24.
- [7] Home Assistant. Concepts and terminology, . URL <https://www.home-assistant.io/getting-started/concepts-terminology/>. Accessed on: 2024-11-23.
- [8] Home Assistant. Entities and domains, . URL https://www.home-assistant.io/docs/configuration/entities_domains/. Accessed on: 2024-11-24.
- [9] Home Assistant. FAQ: This entity does not have a unique ID?, . URL <https://www.home-assistant.io/faq#this-entity-does-not-have-a-unique-id>. Accessed on: 2024-11-24.
- [10] Home Assistant. Homepage, . URL <https://www.home-assistant.io/>. Accessed on: 2024-11-12.
- [11] Home Assistant. Installing a local Assist pipeline. . URL https://www.home-assistant.io/voice_control/voice_remote_local_assistant/. Accessed on: 2024-11-24.
- [12] Home Assistant. Add-ons, . URL <https://www.home-assistant.io/addons/>. Accessed on: 2024-11-22.
- [13] Ashley Carman. The Verge: Amazon will no longer support the Echo Look, encourages owners to recycle theirs. 2020. URL <https://www.theverge.com/2020/5/29/21274805/amazon-echo-look-discontinue-gadget-shopping-recycle-fashion-camera>. Accessed on: 2024-10-30.

- [14] Tom Cava. X10: The Revolutionary Smart Home Device You've Never Heard Of. 2021. URL <https://medium.com/digitalshroud/x10-the-revolutionary-smart-home-device-youve-never-heard-of-48790e272e1f>. Accessed on: 2024-10-30.
- [15] Arindom Chakraborty, Monirul Islam, Fahim Shahriyar, Sharnali Islam, Hasan U. Zaman, and Mehedi Hasan. Smart Home System: A Comprehensive Review. *Journal of Electrical and Computer Engineering*, 2023(1):7616683, 2023. DOI: <https://doi.org/10.1155/2023/7616683>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1155/2023/7616683>. Accessed on: 2024-10-30.
- [16] Control4. About Control4. URL <https://www.control4.com/company>. Accessed on: 2024-11-03.
- [17] DD-WRT. Homepage. URL <https://dd-wrt.com/>. Accessed on: 2024-11-18.
- [18] Docker. What is a Container? URL <https://www.docker.com/resources/what-container/>. Accessed on: 2024-11-19.
- [19] Docker Docs. Docker Compose. URL <https://docs.docker.com/compose/>. Accessed on: 2024-11-20.
- [20] ESPHome. Getting Started with the ESPHome Command Line. . URL https://esphome.io/guides/getting_started_command_line. Accessed on: 2024-11-23.
- [21] ESPHome. Homepage, . URL <https://esphome.io/>. Accessed on: 2024-11-13.
- [22] ESPHome. ESPHome OTA Updates, . URL <https://esphome.io/components/ota/esphome.html>. Accessed on: 2024-11-16.
- [23] ESPHome. Packages, . URL <https://esphome.io/components/packages.html>. Accessed on: 2024-11-23.
- [24] ESPHome. Web Server API, . URL <https://esphome.io/web-api/index.html>. Accessed on: 2024-11-16.
- [25] ESPHome Forum. Servo motor wont work in ESPhome. URL <https://community.home-assistant.io/t/servo-motor-wont-work-in-esphome/429297/3>. Accessed on: 2024-11-23.
- [26] Google. Assistant, . URL <https://assistant.google.com/>. Accessed on: 2024-11-03.
- [27] Google. About Google Home, . URL <https://home.google.com/about-google-home/>. Accessed on: 2024-11-03.
- [28] Ltd Hebei I.T. (Shanghai) Co. TEC1-12703 Datasheet. URL <https://peltiermodules.com/peltier.datasheet/TEC1-12703.pdf>. Accessed on: 2024-11-18.
- [29] IBM. What is a microcontroller? URL <https://www.ibm.com/think/topics/microcontroller>. Accessed on: 2024-11-08.
- [30] Digi International Inc. What Is Zigbee? URL <https://www.digi.com/solutions/by-technology/zigbee-wireless-standard>. Accessed on: 2024-11-02.

- [31] Intel. What Is Bluetooth® Technology? URL <https://www.intel.com/content/www/us/en/products/docs/wireless/what-is-bluetooth.html>. Accessed on: 2024-11-02.
- [32] Frank Joseph. openHAB vs. Home Assistant. URL <https://www.wundertech.net/openhab-vs-home-assistant/>. Accessed on: 2024-11-16.
- [33] Cherie Tan Myles Vendel. Esp32 vs Arduino: The Differences. URL <https://all3dp.com/2/esp32-vs-arduino-differences/>. Accessed on: 2024-11-08.
- [34] OpenHAB. Homepage. URL <https://www.openhab.org/>. Accessed on: 2024-11-12.
- [35] PCMag. Definition of Z-Wave. URL <https://www.pcmag.com/encyclopedia/term/z-wave>. Accessed on: 2024-10-30.
- [36] Inc. QA USA. Disadvantages of Cloud Computing. URL <https://www.qa.com/resources/blog/disadvantages-of-cloud-computing/>. Accessed on: 2024-11-11.
- [37] David Reynolds. ESP32 vs. STM32: How to Choose the Right Microcontrollers. URL <https://www.xecor.com/blog/esp32-vs-stm32>. Accessed on: 2024-11-08.
- [38] IEEE Standards Association (IEEE SA). The Evolution of Wi-Fi Technology and Standards. 2023. URL <https://standards.ieee.org/beyond-standards/the-evolution-of-wi-fi-technology-and-standards/>. Accessed on: 2024-10-30.
- [39] Savant. Company Info. URL <https://www.savant.com/company-info>. Accessed on: 2024-11-03.
- [40] Amazon Web Services. What is Cloud Computing? URL <https://aws.amazon.com/what-is-cloud-computing/>. Accessed on: 2024-11-03.
- [41] Mary E. Shacklett. What is Power over Ethernet? URL <https://www.techtarget.com/searchnetworking/definition/Power-over-Ethernet>. Accessed on: 2024-11-02.
- [42] Muhammad Shafiq, Zhaoquan Gu, Omar Cheikhrouhou, Wajdi Alhakami, and Habib Hamam. The Rise of “Internet of Things”: Review and Open Research Issues Related to Detection and Prevention of IoT-Based Security Attacks. *Wireless Communications and Mobile Computing*, 2022(1):8669348, 2022. URL <https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/8669348>. Accessed on: 2024-11-02.
- [43] Alan Stafford. First Look: Insteon’s Easier Home Automation. URL <https://www.washingtonpost.com/wp-dyn/content/article/2005/09/02/AR2005090200875.html>. Accessed on: 2024-11-03.
- [44] Tasmota. Homepage, . URL <https://tasmota.github.io/>. Accessed on: 2024-11-16.
- [45] Tasmota. Web Server API, . URL <https://tasmota.github.io/docs/MQTT>. Accessed on: 2024-11-16.
- [46] Tasmota. Upgrade using webUI, . URL <https://tasmota.github.io/docs/Upgrading/#upgrade-using-webui>. Accessed on: 2024-11-16.
- [47] Harshita Tewari. 11 Best Free Virtual Machine Software in 2024. URL <https://learn.g2.com/free-virtual-machine-software>. Accessed on: 2024-11-23.

- [48] UpKeep. What is the difference between Industry 3.0 and Industry 4.0? URL <https://upkeep.com/learning/industry-3-0-vs-industry-4-0/>. Accessed on: 2024-10-30.
- [49] VMware. What is a Virtual Machine? URL <https://www.vmware.com/topics/virtual-machine>. Accessed on: 2024-11-23.
- [50] YAML. Homepage. URL <https://yaml.org/>. Accessed on: 2024-11-23.
- [51] Kinza Yasar. What is a Smart Home? Everything You Need to Know | Definition from TechTarget. URL <https://www.techtarget.com/iotagenda/definition/smart-home-or-building>. Accessed on: 2024-11-11.