WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
MASTER STUDY PROGRAM: Software Engineering

MASTER THESIS

SUPERVISOR:
Asist. univ. dr. Sebastian Stefaniga

GRADUATE:
Bobosila Victor

TIMIŞOARA
2023

WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
MASTER STUDY PROGRAM: Software Engineering

Home Assistant

SUPERVISOR:
Asist. univ. dr. Sebastian Stefaniga

GRADUATE:
Bobosila Victor

TIMIŞOARA
2023

Abstract

Automating tasks originally performed manually has always been a definitive goal for humanity. From the early water clock devised in the Ptolemaic Egypt, to fully automated factories and markets found in the modern digital era. One area of interest for automation has always been the notion of smart homes, which assist the people occupying them with various mundane tasks. This work presents the proposal of introducing an assistant-like technology that allows users to actively and passively automate and optimize one's interactions with different household appliances. At the base of this proposal, Internet of Things (IoT) devices, such as temperature, humidity, presence and other sensors, together with smart lights, switches and plugs, serve a crucial role. These smart devices help collect necessary data which is used by another major element of this work, a Raspberry Pi. The Raspberry Pi makes use of this data to both present it in a user friendly manner, but also to make decisions regarding different optimizations, such as lowering the temperature and turning off lights in a passive manner, with the help of machine learning algorithms.

Keywords: Internet of Things, Smart Home, Optimization, Assistant, Raspberry Pi, Smart Devices

Contents

Chapter 1

Introduction

Home automation systems, although seemingly recent, have been a topic of discussion since 1966, when the first true home automation device was created. A 800 lb computer which allowed the control of temperature and different house hold appliances, named ECHO IV [25], although extremely impractical and expensive, offered an insight to what is possible.

Now days, with the increase of computational power, the decrease of physical size, and the ever growing hope of reducing manual tasks in form of human labor, a great deal of research has been dedicated to the usage of Internet of Things (IoT) devices for home automation purposes. The term Internet of Things was created by Kevin Ashton in 1999, which described a method of allowing computers to know more about data, to help understand, track and count everything, therefore improving our lives by reducing costs, losses and waste [7]. These IoT devices act as a perception layer, that collect data from the surrounding environment by "sensing it", interact and communicate with each other, and react to the environment by triggering actions and creating services based on the data collected, without any human intervention [28]. Further advancements in machine learning also helped with the development and research of home automation applications. By incorporating different prediction, voice recognition and face recognition algorithms, the notion of "assistants" began to rise.

Therefore, these home assistants which help automate one's house, started being commercially available. Amazon Alexa, Google Assistant, Siri, Cortana and various other solutions have been devised to enable a user to automate different household appliances, lights, temperature and other features. That being said, most introduce the notion of "vendor lock-in" [27], where a company forces their users to only use products from within the said company. Different types of sensors or lights that aren't proprietary are not supported, which can introduce a certain level of frustration. Because of this, different non-commercial solutions began to surface. To this day most of these projects use portable platforms such as the Raspberry Pi, commonly know as a "hub", to house the business logic necessary to interact with the IoT devices. The communication between the smart devices and the hub is often realised by protocols such as Zigbee, Z-Wave, Bluetooth and WiFi, and the interface with the user is usually represented by a simple web application, that offers control over the devices and the data they sense.

Although both commercial and non-commercial solutions offer the possibility of automating most tasks, certain limitations can be observed. The usual pattern of

limitations can be defined by the following properties:

- Commercially available home automation and assistant services introduce vendor-lock in issues, minimal user interfaces replaced by voice recognition, and a mostly **passive** way of automating different tasks, usually with the help of machine learning algorithms.

- Non-commercial, open-source solutions serve a user driven, **active** way of automating tasks, by extensive configuration of the smart devices, through a traditional, web application user interface.

Considering the points mentioned above, this proposal aims to combine both the active and passive elements found in most home automation projects and products, to allow an user to actively automate different tasks around the house, and passively optimize elements such as power and temperature, with the end goal of achieving a energy efficient, automated house, therefore creating a Home Assistant.

1.1 Target Group

This work is addressed to all IoT and home automation enthusiasts, as well as any new Masters students hoping to begin learning or further understand the importance of automation and the extent of possibilities introduced by IoT devices. It is assumed that the reader is well versed in programming languages such as TypeScript and Python, has a basic understanding of networking elements such as Websockets and understands communication protocols such as MQTT and Zigbee.

1.2 Motivation

True automation of a house has always been a topic of interest for me. The extensive improvements regarding quality of life that a product of this nature can bring, are undeniable. That being said, the experience I had with commercial products were disappointing, lacking in options and fairly expensive, while the open-source projects were of poor quality and lacking features expected as of today, such as different ways of passively optimizing with machine learning algorithms. These issues, together with the current geopolitical climate, which threatens the lively hood of many people, brought the motivation to study the topic of house automation with the purpose of optimizing one's life, both from an energy perspective, by reducing costs with the help of automating temperature and power consumption, but also from a time perspective, by allowing the user to schedule and automate different daily tasks.

1.3 Research method

To find out weather this proposal has any merit, validating the quality of life improvements is paramount. This will be done by comparing costs with and without the proposed work, but also by an objective personal satisfaction factor, measured by multiple individuals. To do so, the project must be implemented inside multiple

users houses, and given time to acclimate to the product. Areas of interest that should be measured can be listed as:

- **Ease of use**: How easy is it to start automating different tasks? Is the graphical interface user friendly? Can a user easily access everything they need?
- **Speed**: How fast can a user collect the data they need? Is the application fast enough to keep up with their needs?
- **IoT devices**: Does the product support all the devices the user needs? Are they easy to discover and access?
- **Cost reduction**: How much money is saved by using the assistant? Is it worth it to use the product from this perspective?
- **Effort reduction**: How much does the product reduce the users daily efforts? Does the quality of life improve?
- **Model performance**: Objective review of the performance of the models used for optimizing energy efficiency.

1.4 Structure

From this point on wards, the presented work will adhere to the following structure:

Related Work A discussion of already completed previous work. This section will present an in depth analysis of related literature, as well as means of further developing or expanding the solutions presented. The information collected in this section is paramount to the development of the proposal, since it will contain the basic foundations of this work.

Proposed System In depth presentation of the proposed system of this work. The architecture, design, data collection, features descriptions, usage of machine learning for optimization together with software and hardware requirements will be discussed and explained. This section is the most relevant piece of information in this work, since all the data collected from here will be used for further discussions later on.

Results This section presents and elaborates all the information that resulted from the previous topic. Any evaluations and measurements will be discussed and compared with any related work used for this proposal.

Conclusion Final section, containing the last words explaining whether or not the proposed system and results were satisfactory.

Chapter 2

Related Work

2.1 Temperature prediction and monitoring using IoT devices

Different types of research have been conducted regarding temperature and weather prediction. These can be split in 2 categories:

- Indoor temperature prediction and monitoring.
- Outdoor temperature and weather prediction and monitoring.

Most extensive research regarding outdoor weather forecasting is in regard to climate study. Manzhu Yu et al.[30] describes the usage of Recurrent neural networks(RNN) such as Long Short-Term Memory(LSTM), for predicting surface temperature forecast in an urban environment such as New York. Using data from IoT devices gathered by GeoTab, which collects air temperature data from vehicle mounted sensors, together with archived weather station data, a LSTM model is trained. It is described that an LSTM model is a natural fit for this type of problem, since it can learn for a longer period of time, compared to simple RNN. Further so, it is explained that the described model is relatively insensitive to extreme outliers that can appear from strong weather fluctuations. Similar research using the same type of LSTM model can be found in most other outdoor temperature and weather prediction research [8] [29]. Conclusive evidence of the effectiveness of this model is described by Hewage et al.[12] which describes the comparison of a data driven weather forecasting model such as the previously mentioned Long Short-Term Memory, with the already existing weather forecasting Numerical Weather Prediction (NWP) model. Although this data doesn't make use of IoT devices, but instead of weather station data which contains important parameters such as temperature, humidity and wind speed. Results show that the model outperformed the NWP model for up to 12h. Even though the data collected is not directly recorded from IoT devices, this is a non-issue, since different sensors can collect the data for each parameter of this model, therefore the pieces of research mentioned above could be translated into the context of this work.

Indoor temperature prediction and monitoring follows similar methodology to the outdoor counterpart. For example Debayan et al.[20] describes a very similar methodology to Manzhu Yu, but on a much smaller scale, a residential building

in Granada Spain. It makes use of very similar data attributes, from indoor temperature, outdoor temperature, solar radiance and others to train different traditional machine learning algorithms such as Random Forest, Support Vector Machine(SVM) and Neural Networks. The results of this work showcase fairly similar performances, with approximate R^2 values of 0.98, 0.88 and 0.92 for Random Forest, SVM and Neural Networks respectively. Although close results, the clear and distinct winner is the Random Forest model. In a very similar manner to Debayan, Moreno et al.[18] showcase a similar building in Granada Spain, but with some other models to be researched. The focus of this work are the following models:

- Multi-Layer Perceptron (MLP)
- Support Vector MAchines with Radial Basis Function Kernel (SVM)
- Gaussian Process with Radial Basis Function Kernel
- Bayesian Regularized Neural Networks (BRNN)

In this case, the BRNN model is the distinct winner, with an average R^2 value of 0.97.

For all of these related pieces of literature, we can see a clear trend of using machine learning/deep learning techniques for the prediction of temperature. Whilst the type of research is divided into outdoor and indoor experiments, due to the extent of IoT sensors we can gather relatively similar data, therefore its possible to use the same top 3 models resulted from the research presented:

- LSTM - Seen in both outdoor and indoor studies, performed very well
- Random Forest - Mostly seen in indoor studies, with great performance, which serves the context of this work very well

From these findings, it should be expected that these 2 models are worth comparing to see which performs better in the context of this proposal. Furthermore, this notion of predicting temperature can be extended to other household appliances as well, such as smart lights or smart plugs. The prediction model could predict when these devices are usually activated, and turn them on, in a similar fashion with the temperature.

2.2 Home automation in the context of smart homes

Several pieces of literature categorize home automation systems in the following groups [26][9]:

- **Bluetooth Based**
- **GSM Based**
- **Zigbee Based**

2.2.1 GSM Based Home Automation Systems

Previous works on this topic describe similar methodologies. In general, most architectures contain the following components as [17][24][19][10]:

1. **Mobile Phone**. This component serves as the human-machine interface, and allows the user to control and monitor household devices.

2. **GSM Modem**. Used as a communication interface. This module is connected to the Mobile Phone and a controller interface (usually a RS232). It can receive instructions from the mobile phone and send them to the controller interface to perform different actions.

3. **RS232 interface**. Interface which defines signals connecting between a data terminal equipment(in this context usually a micro controller) and data communication equipment(usually a modem).

4. **Controller**. Houses all the logic necessary to perform desired actions, such as turning on a light bulb, and its responsible for sending and receiving data to and from devices. This component seems to vary heavily, ranging from Arduinos, PIC micro controllers and FPGA micro controllers.

5. **Devices**. Endpoint of this general architecture. These can range from alarm systems, lights, fans, music systems and more. They are directed by the Controller and send data back to the same component.

This type of home automation system works by using the mobile phone to send SMS messages, which are translated by the controller and used to perform tasks, such as turning on all light bulbs, or starting and stopping fans. The user can also receive messages containing the status of a certain device. This is a simple architecture and although fast to implement it lacks in scalability and the usage of SMS messaging can become a hindrance from the user experience point of view.

2.2.2 Bluetooth Based Home Automation Systems

Research in this topic follows a very similar architecture to the previously described GSM based model. Works such as [16][22][13], describe architecture which contains:

1. **Mobile Phone with Android Application**. This serves as the user interface, based on an Android application. Compared to the previously GSM based SMS interface, this proves to be a huge improvement in user experience, allowing the user to better interact with the devices. This component also communicates with a Bluetooth module to send and receive information.

2. **Bluetooth module**. Component which takes the role of a communication interface. It is connected to the Controller and the Mobile Phone, in a very similar fashion to the GSM modem component. It seems that usually a HC-05 module is used.

3. **Controller**. Similar to the GSM model Controller, although in this case most works make use of an Arduino Uno. A slight difference is that this component does not communicate directly with devices, but instead interacts with a relay switch.

4. **Relay switch**. This component receives information from the controller, and sends tasks to the devices.

5. **Devices**. Endpoint of this architecture, similar to the GSM based model.

Although not too different from the GSM model, the quality of life improvements that come from the usage of an Android application are noteworthy. It is important to mention that even though mobile applications have been used in the works presented, web, desktop and cloud applications could also be implemented in stead of mobile.

2.2.3 Zigbee Based Home Automation Systems

In [11], Zigbee is defined as a "mesh-networking standard based on IEEE 802.15.4 radio technology targeted at industrial control and monitoring, building and home automation, embedded sensing, and energy system automation". To expand on this Jin-Shyan et al.[15] further elaborates the types of devices that can participate in such a network, describing them as:

Full-function devices This device can operate in three modes: PAN (personal area network) Coordinator, Coordinator, or a device. FFDs can talk to RFDs and other FFDs. They can provide synchronization services to other devices or other coordinator, but only one of these can be the overall PAN coordinator.

Reduced-function devices This device device is intended for applications that are simple, such as a light switches or passive infrared sensors, since they do not have a need to send large amounts of data. RFDs can only communicate with other RFDs, and can only associate itself with one FFD.

The same work goes on to compare different wireless protocols including Zigbee, WiFi, Bluetooth and UWB (Ultra-Wide Band) from different perspective such as radio channels, coexistence mechanisms, network size, security, transmission size and data coding efficiency. From this study, it is observed that Bluetooth and Zigbee are suitable for low data rate applications with limited battery power whilst UWB and WiFi is better suited for high data rate implementations.

That being said, it is no surprise then that most home automation applications make use of Zigbee and Bluetooth devices predominantly, but in special cases WiFi communication is necessary for more demanding devices. Examples of such applications are presented in [21][23][14], which follow a similar architectural pattern to the Bluetooth and GSM models.

1. **Application Layer**. This layer contains the application with which the user interacts. It should facilitate the possibility to detect, monitor and control any devices connected to the application.

2. **Transport and Network Layer**. Facilitates a way of communication between the Data Link Layer and the application layer. This can be HTTP, Websockets and any other means of communication.

3. **Data Link Layer**. This component contains the **device manager**, usually a micro controller which uses a zigbee coordinator to detect any Zigbee devices

in a certain range. It also contains the **IoT gateway router** which sends information from all devices detected to the transport layer.

4. **Physical Layer**. Contains all the IoT devices.

Although similar in concept to the other models described, a fair bit of complexity has been added to this type of home automation system. It is worth to mention that this architecture can be scaled to other types of devices, such as Bluetooth or WiFi through additional hardware upgrades, such as Bluetooth modules (HC-05) or WiFi modules (ESP8266WiFi).

Chapter 3

Proposed System

The proposed system for this work follows similar patterns to the previously discussed projects. For this specific system, a Zigbee approach is desired due to the low energy cost and abundance of Zigbee compatible devices.

3.1 Architecture and Design

3.1.1 General System Overview

Figure 3.1: General System Architecture Overview

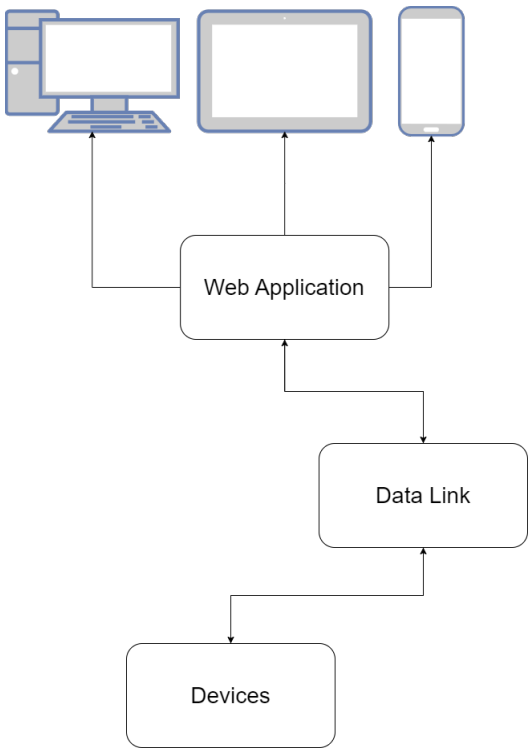


Figure 3.1 showcases how the general system functions. The architecture contains devices which connect to the web application, offering the user the possibility to interact with the application, the previously mentioned data link layer which contains the device discovery and management system, and the smart devices which

are exposed to the user to control and monitor. It is clear that the architecture follows traditional approaches regarding home automation design. That being said the main differences can be found in the specific layers of the architecture.

Device Layer

The device layer consists of all the IoT devices which were placed in a singular room.

Figure 3.2: Architecture of target room

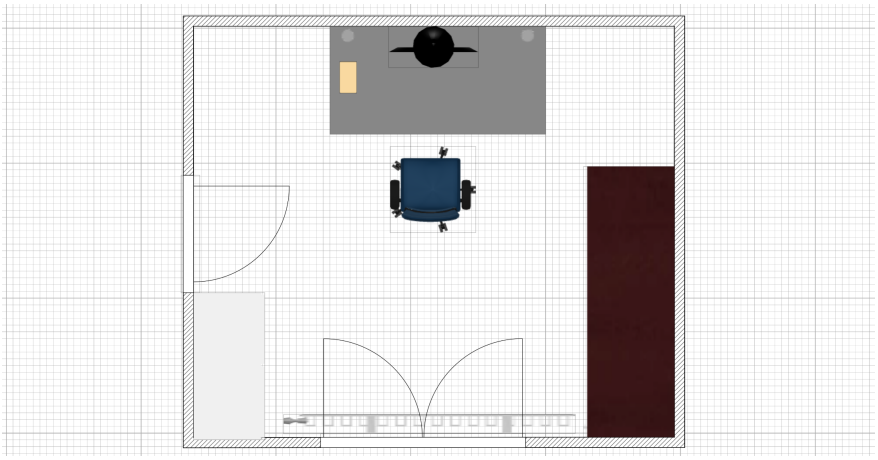


Figure 3.2 represents the architecture of room which is used as the target environment for this work. The characteristics of the room are as following:

- **368** centimeters in length
- **316** centimeters in width
- A total of **11,7024** m^2

Figure 3.3: 3D render of the room

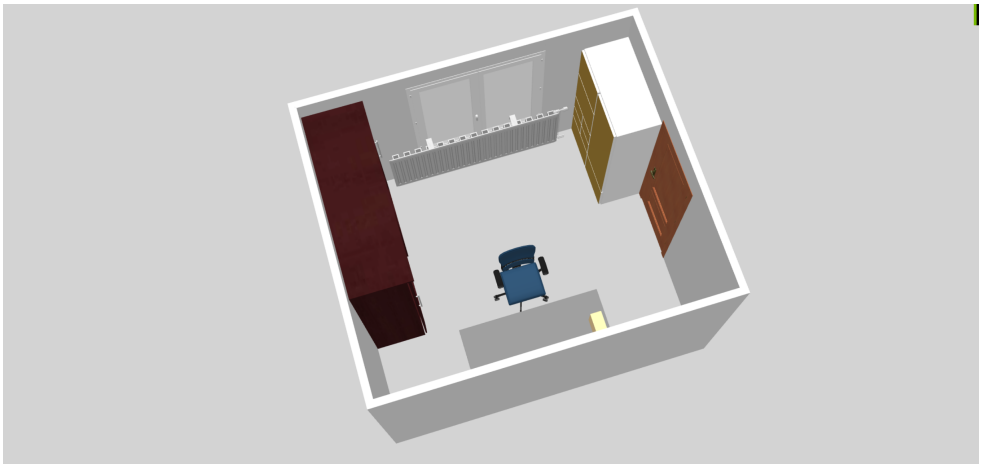


Figure 3.3 represent a 3d rendering of the base architecture. Here we can better see the main object of importance which is the radiator. The radiator is controlled

by a HY368 Smart Home Tuya smart radiator valve. The data used for providing temperature information is extracted from two SNZB-02 Temperature and humidity sensors. In total there are 9 devices scattered around the room:

- **1. TRADFRI on/off switch.**
  - Location of this device is on the office table
- **2. LCA006 Hue white and color ambiance E27 light bulb**
  - Location of this device is in the top right corner of the room
- **3. MS01 SNZB-03 Motion sensor**
  - Location of this device is on the top of the bottom left wardrobe.
- **4. Two SNZB-02 Temperature and humidity sensors**
  - Location of these devices are the following:
    - One next to the window of the room
    - One next to the computer on the table
- **5. SNZB-04 Contact sensor**
  - Location of this device is on the door.
- **6. GZCGQ01LM MiJia light intensity sensor**
  - Location of this device is on the office table.
- **7. HY368 Smart Home Tuya smart radiator valve**
  - Location of this device is connected to the office radiator.
- **8. Livarno Lux E27 bulb RGB**
  - Location of this device is connected to bottom left corner of the room.

Data Link

Figure 3.4: Data Link Architecture

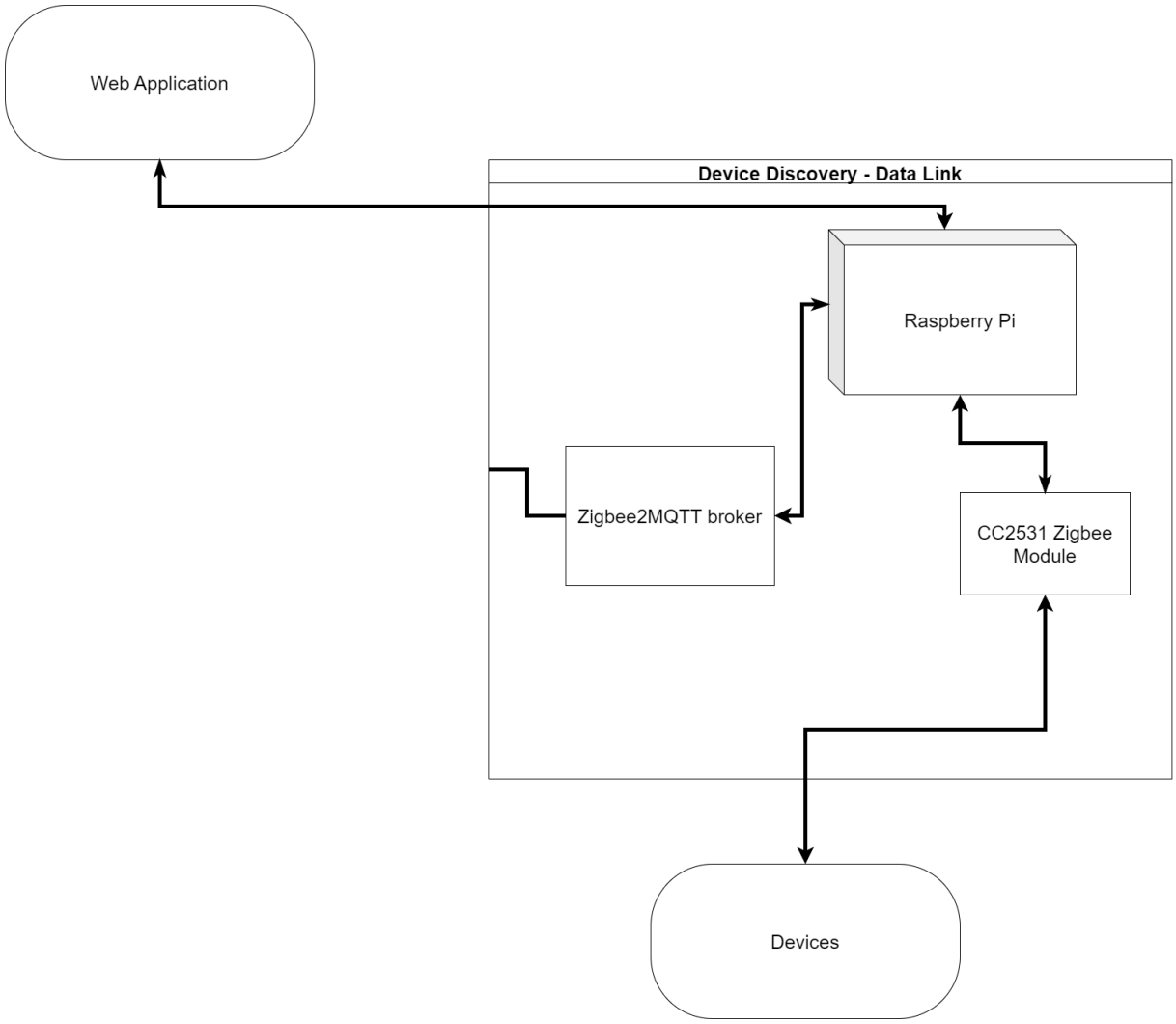
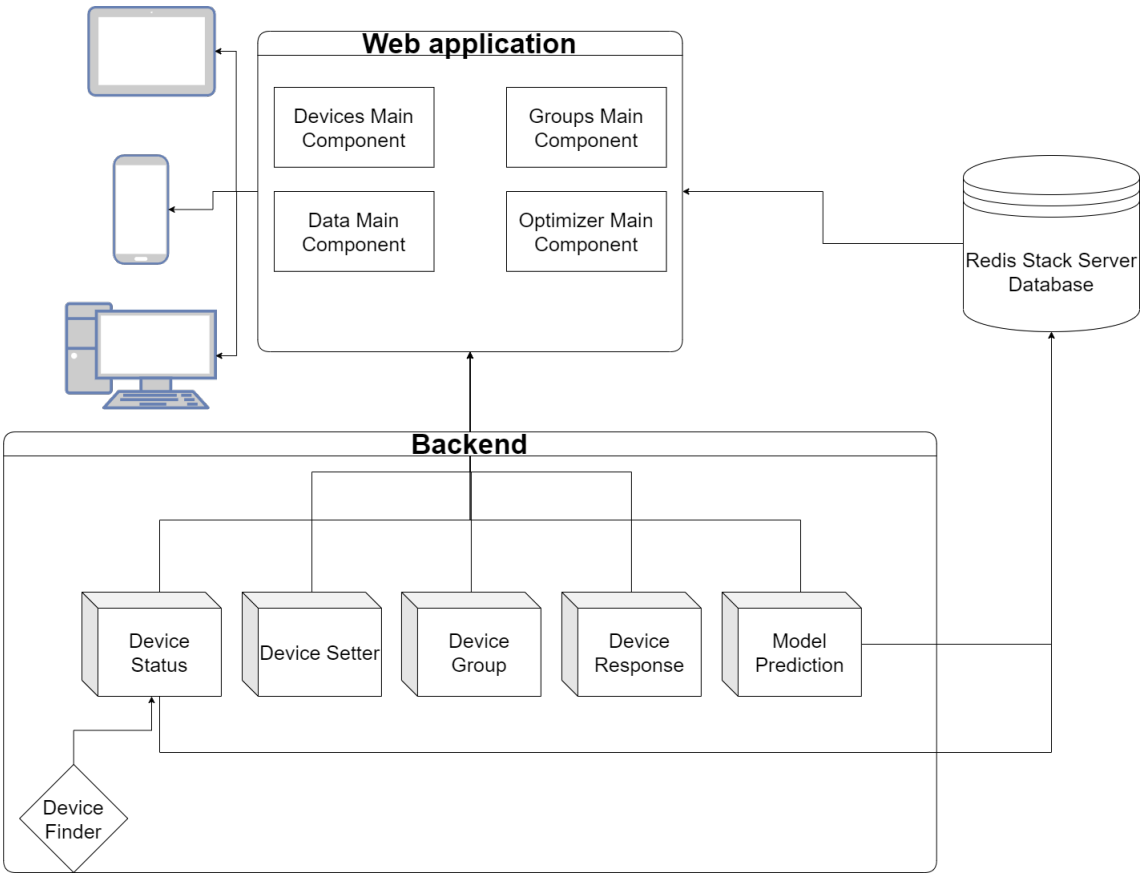


Figure 3.4 showcases the device link layer, which uses a Raspberry Pi at the base, to facilitate communications between the user and the smart devices. To manage this, a CC2531 Zigbee dongle is used as an interface with the Zigbee devices. Although this facilitates a communication layer, a gateway is necessary to avoid using the device's manufacturer gateway, hence avoiding vendor lock-in issues. For this, an open source project called zigbee2mqtt [4] is used. This service connects Zigbee networks to MQTT networks, which are easier to handle, by being a publish/subscribe type protocol.

The Raspberry Pi configuration is a Model 4 B with a Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz processor and 8 gigabytes of LPDDR4-3200 SDRAM. This choice is important since the machine learning model and the web application will run on this device, so the more computing power and storage the better. The operating system used for this proposed solution is Ubuntu Desktop 22.04 LTS, since it provides the best compatibility for the packages and modules required to build this project.

Web Application

Figure 3.5: Web Application Overall Architecture



To be able to offer the user a means to interact with the smart devices in a user friendly manner, a web application has been devised, as seen in Figure 3.5. The purpose of this application is to allow the user to find, control and manage all devices detected by the data link layer. It is expected that a user can group devices, bind them together(e.g bind a light bulb to a smart button), change any values that can be changed, and visualize the data transmitted by the devices. To meet these expectations, the backend of the proposed solution implements 4 dedicated services for device interactions:

- **Device Status**
  - This service receives information regarding devices from another device named Device Finder. It then uses this device information to gather state data about these devices (e.g temperature data, humidity data, brightness value of light bulb etc.)
- **Device Setter**
  - This service allows the user to send action requests to all the IoT devices which allow actions, or directly to the bridge. Here the user can change

values such as the brightness of a light bulb, action of a switch, temperature of the radiator valve, create and delete new groups as well as bind devices together

- **Device Group**
  - This service checks for device groups
- **Device Response**
  - This device checks for responses from previous actions that the user made (e.g when binding two devices together a response will be given, and the project can use this to notify the user.)

In the IoT context, devices generate a large amount of data in a fast manner. Because of this, the database of choice is Redis, which is a in-memory data store. It can handle a large amount of data and it can store it without delay, which is important since the proposed solution uses time series data. Any delays can affect the time series and can therefore impact the accuracy of the data. It is important to mention that persistence was enabled, since in it's default state, the data only lives in-memory, and a reboot would lose all the stored data.

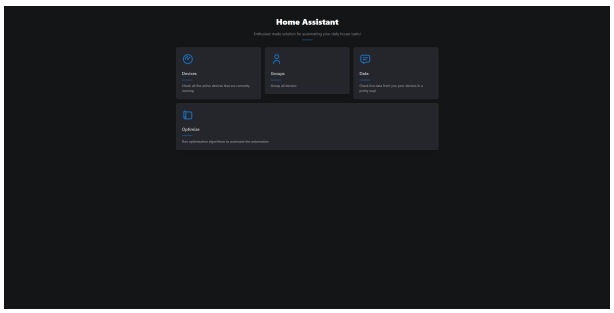
As previously mentioned, a passive means of automation has to be implemented. For this purpose, a separate service entitled Model prediction is proposed. It gathers past data from the database, and uses it to create predictions, which in turn are used to control the radiator valve, and at the same time they are stored back into the database to offer a timeline of past predictions. With this service temperature regulation can be achieved without any user input, but at the same time a safety feature must be implemented in case of prediction failure (e.g limit the maximum temperature allowed to be predicted to 30 degrees Celsius)

3.1.2 Design

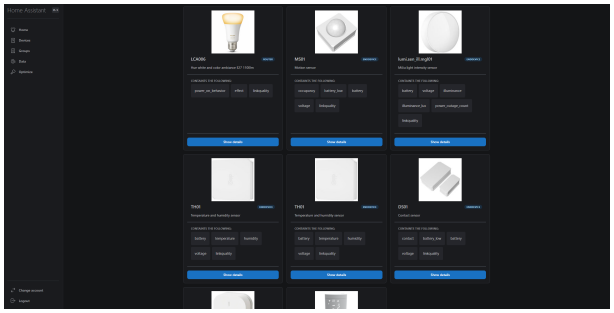
Since user experience is an important topic in this work, the proposed system uses different techniques to achieve this goal:

- **Usage of grids**. This allows for easy organization of components and structures in the layout of the page, offering a easier navigation experience for the user
- **Usage of cards**. Offers a better alternative to using lists for ordering and storing data by compartmentalizing relevant information in their own structures
- **Responsiveness**. This improves the accessibility of this work by allowing the web application to function on different devices with different screen sizes.
- **Real time**. Allowing for real time data retrieval and real time device interactions is incredibly important in the context of IoT development. This way the user can see in real time the changes made by the application.
- **Data visualization**. This gives the user the option to visualize the past and current data from the devices in the network.

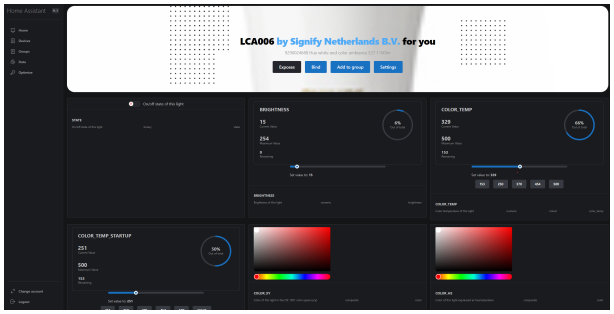
- **Minimalist approach**. To avoid cluttering the pages with unnecessary data, a minimalist approach is used. Data is only rendered when needed, and due to the easy organization of components with the help of grids and cards, it is possible to create the minimum interactions needed to collect or send information to devices



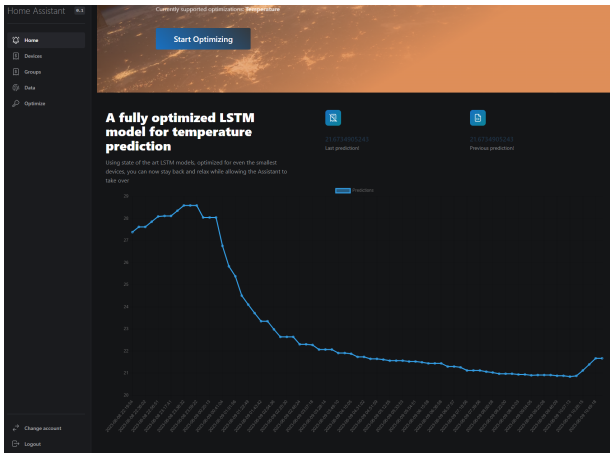
(a) Minimalist approach



(b) Grid system



(c) Example of cards



(d) Plotting

Figure 3.6: Example of the usage of design techniques mentioned

Figure 3.6 showcases the different techniques used for maximizing the user experience. It is important to note that another benefit of the minimalist approach also helps the performance of the application since there are less components to render at the same time on a single page. This is crucial since the project runs on limited hardware and it is important to save as much computing power for predictions and data processing.

3.2 Implementation

In this section, the implementation for the proposed system will be described. As mentioned in 3.1.1, the proposed solution has 3 architectural layers:

- **Device** layer
- **Data Link** layer
- **Web Application** layer

Since no real work is required on the Device layer, the rest of this section will be focusing on the Data Link and Web application layers

3.2.1 Data Link Implementation

Figure 3.7: Raspberry PI with Zigbee sniffer

Figure 3.7 shows the hardware components necessary for the Data Link layer. The Raspberry Pi model 4B can be observer together with the CC2531 Zigbee dongle used to communicate with IoT devices in the network. A mouse, keyboard and HDMI cable are also connected since it is necessary to setup the Zigbee2MQTT software on the device. There are multiple ways a user can set up this piece of software, such as a docker container or a python virtual environment, but for the purposes of this work, a simple Linux installation is preferred. The steps required to install Zigbee2MQTT are well described in their official website guide [5], but to summarize, the procedure is as follows:

- Determine the location of the CC2531 adapter (usually on /dev/ttyACM0)
- Cloning and installing the software
- Checking the installation was successful
- Start the software with the Node Package Manager (NPM)

- *Optional*: Configure the software as a service. This is usually preferred since in the case of a reboot or power outage, the service will automatically restart.

Once Zigbee2MQTT was installed, pairing the IoT devices with the Zigbee module is necessary. Each device has an unique way of pairing, so this work will only describe the pairing methodology for the 9 selected devices:

- **1. TRADFRI on/off switch.**
  - Unscrewing the bottom latch, a button can be observerd. If pressed 4 times a red light will start flashing on the top of the device, meaning that the switch is in pairing mode.
- **2. LCA006 Hue white and color ambiance E27 light bulb**
  - A sequence of turning on/off for 5 times, with a waiting time of 5 seconds in between each action will make the device flash, meaning that it is currently in pairing mode.
- **3. MS01 SNZB-03 Motion sensor**
  - A small hole can be observed on the side of the device. A pin or a toothpick can be used to access the button which resides in the hole. Holding it pressed for 5 seconds will enter the device in pairing mode.
- **4. Two SNZB-02 Temperature and humidity sensors**
  - Location of these devices are the following:
    - By holding the button on the side of the device for 5 seconds will enter the sensor in pairing mode.
- **5. SNZB-04 Contact sensor**
  - A small hole can be observed on the side of the device. A pin or a toothpick can be used to access the button which resides in the hole. Holding it pressed for 5 seconds will enter the device in pairing mode
- **6. GZCGQ01LM MiJia light intensity sensor**
  - By holding the button on the side of the device for 5 seconds will enter the sensor in pairing mode.
- **7. HY368 Smart Home Tuya smart radiator valve**
  - Using the touch screen interface of the smart radiator valve, the pairing mode screen must be selected. Afterwards, by holding the home button for 5 seconds, the device will start pairing.
- **8. Livarno Lux E27 bulb RGB**
  - A sequence of turning on/off for 5 times, with a waiting time of 5 seconds in between each action will make the device flash, meaning that it is currently in pairing mode.

Figure 3.8: Zigbee service outputting data

Figure 3.8 represents the expected output of the devices after the pairing is done.

3.2.2 Web application

The description of the web application implementation can be split into two categories:

- **Active automation.** This category will contain any component in which the user will actively interact with devices.
- **Passive automation.** This category is represented by the creation, training and deployment of the machine learning algorithm used to passively regulate the indoor temperature of the room, therefore excluding any user input

3.2.3 Interactivity with devices

To implement the active automation component of the proposed solution, it is necessary to first understand the how a user can interact with IoT devices trough the MQTT protocol and the Zigbee2MQTT software.

Figure 3.9: Example of subscribing to topic trough MQTT

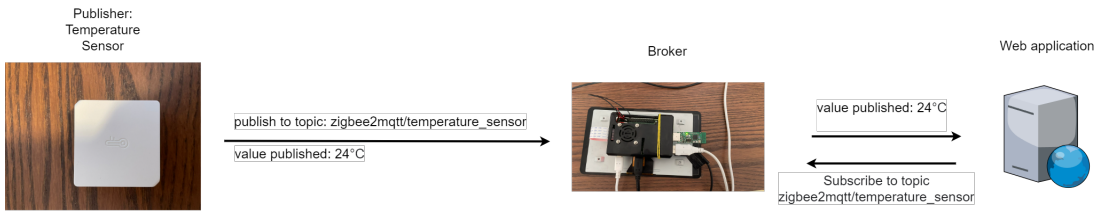


Figure 3.9 exemplifies the action of subscribing to a specific topic and receiving data from an IoT device. To further elaborate, the diagram showcases a temperature sensor which constantly publishes data to the **zigbee2mqtt/temperature_sensor** topic inside the broker. This in turn can then be accessed by different MQTT clients, such as a web application, which can subscribe to the topic and receive the real time information from the IoT device, in this case the information is the temperature at a specific timestamp.

There are a multitude of subscribable topics available, although the ones of importance are the following:

- **zigbee2mqtt/*device_name***, where device_name represents the name of the device. This topic contains information regarding the current state of the device. For example for a temperature sensor, one could expect to receive information regarding the temperature, battery level and signal strength.

- **zigbee2mqtt/bridge/devices**. This topic contains information regarding all devices connected, but not the state of the device. This information is composed of the device name, ieee address, device description, model id, manufacturer, list of exposed signals (e.g temperature, battery, humidity, etc) and more relevant device information.

- **zigbee2mqtt/bridge/groups**. This topic has general information about any zigbee groups made. In general these groups represent a collection of devices.

- **zigbee2mqtt/bridge/response/device/*bind/unbind***. This topic offers responses after binding/unbinding actions. This is useful for notifying the user on whether the binding/unbind went well or errored out.

Figure 3.10: Example of publishing to topic trough MQTT

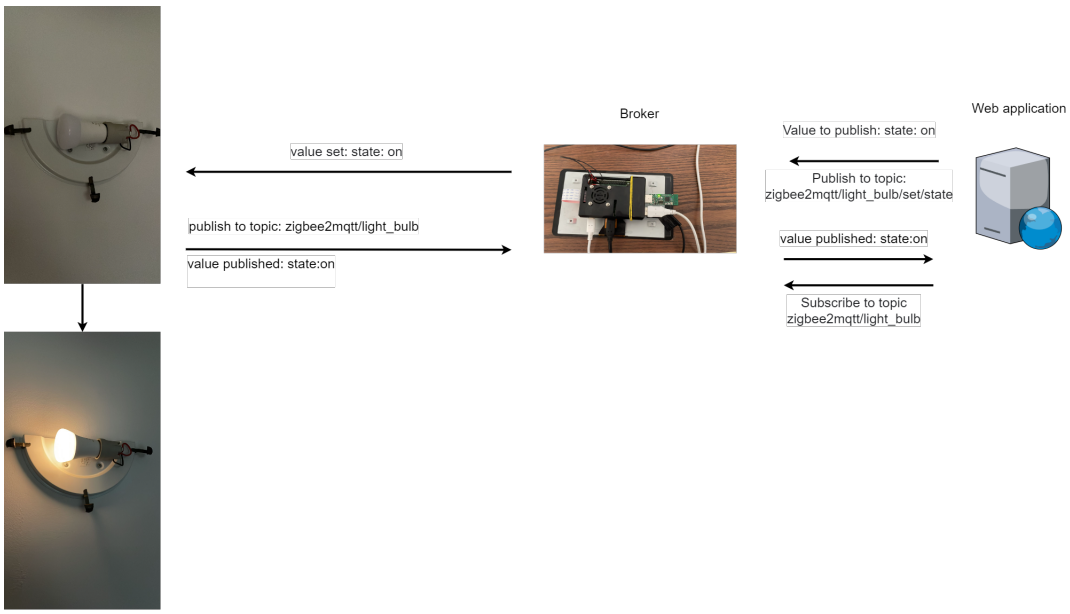


Figure 3.10 showcases the action of publishing user information to devices. From the web application the user can set the state of the device to **on** by publishing this message to the *zigbee2mqtt/light_bulb/set/state* topic. This information is then sent to the device by the broker, and the light bulb will turn on, returning a message confirming the new state of the device.

To understand what can be published and when, it is important to further discuss the exposed signals of different devices. Each device has a list of exposed signals, which can be categorized as such:

- **Generic**. It will always contain the name and type of the property exposed. The types can be the following:

  - **Numeric**. Exposes a numeric value, and can optionally contain the following signals:
    * *value_max* (the maximum value possible)
    * *value_min* (the minimum value possible)
    * *value_step* (the step with which the value can be changed)
    * *presets* (a preset of predefined values)
  - **Enum**. It always exposes a signal named **value** which is a list of predefined values
  - **Binary**. It always exposes two signals named **value_on** and **value_off**. Can optionally contain:
    * *value_toggle*
  - **Text**. Indicates a device exposes a textual value

- **Composite**. Composite combines the above generic types in a signal named features. For example a light could have a composite property named color, which contains two generic numeric types.

- **Specific**. Specific types use both composite and generic types to create well defined structures. These can be the following:
  - **Light**
  - **Climate**
  - **Switch**
  - **Fan**
  - **Lock**
  - **Cover**

Figure 3.11: Example of possible exposed signals

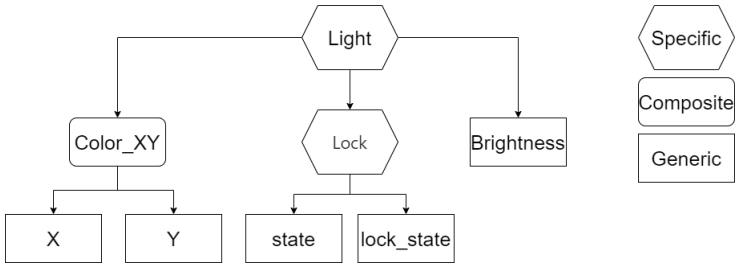


Figure 3.11 exemplifies a possible configuration for a light device, using the types of exposed signals defined earlier.

3.2.4 Active automation

Backend

The first step in the implementation of the active component of the solution, is to create the backend services which can send the information to our frontend components. For this MQTT clients are created using the python library **paho_mqtt**. These clients connect to the Zigbee2MQTT broker and implement the following callbacks:

- **on_message**. Called when a message is received,
- **on_connect**. Called when the client connects
- **on_disconnect**. Called when the client disconnects
- **on_subscribe**. Called when a client subscribes to a topic
- **on_publish**. Called when a client published a message to a topic.

There are six total clients which were created for the implementation of the solution:

- **DeviceFinder**. After this client connects to the broker, it subscribes to the "zigbee2mqtt/bridge/devices" topic and collects all the devices in the network. This client then disconnects and immediately connects back. This is done in order to refresh the information, since the broker doesn't auto provide the data necessary.
- **DeivceStatus**. This client uses the data from "DeviceFinder" to identify the names of all devices. It then uses these names to subscribe to each device collecting the states of these devices. These states are then appended appended to the device information collected by "DeviceFinder", therefore collecting all the data necessary for a device.
- **DeviceGroupFinder**. This client behaves in the same manner as "DeviceFinder", except the topic this client subscribes to is "zigbee2mqtt/bridge/groups" and it never closes.
- **DeviceResponseBind**. This client subscribes to "zigbee2mqtt/bridge/response/device/bind" and listens to any responses coming from the devices
- **DeviceResponseUnBind**. Similar to the bind counterpart, except the topic is "zigbee2mqtt/bridge/response/device/unbind"
- **DeviceSetter**. This client handles all the user interactions by implementing specific publish actions such as:
  - PublishCustom. This function receives the device name, exposed signal that the user wants to change, and the message as parameters (e.g self.client.publish(f"zigbee2mqtt/{device}/set/{topic_raw}", message))

– PublishNewGroup. This function receives the action which will be done (e.g add/remove) and the name of the group as the message (e.g self.client.publish(f"zigbee2mqtt/bridge/request/group/{topic_raw}", message_sent_string))

– PublishCustomBind. This function receives the action which will be done(e.g bind/unbind), the source device and the target device of the binding. (e.g self.client.publish(f"zigbee2mqtt/bridge/request/device/{choice}", message_sent_string))

These MQTT clients are then introduced into web sockets which will continuously serve the data to our web application frontend and at the same time store the data into the Redis Database. In this step keys are created for each device with each property (e.g "temp_sensor:temperature", "temp_sensor:humidity" etc), and the value for each key is a timestamp and the property value.

```python
async def handler(websocket):
    async for message in websocket:
        print(message)
        if message.split()[0][1:] == "add_group":
            DeviceSetter().publishCustomNewGroup(
                message.split()[1], message.split()[2][:-1]
            )
        elif message.split()[0][1:] == "remove_group":
            DeviceSetter().publishCustomDeleteGroup(
                message.split()[1], message.split()[2][:-1]
            )
        else:
            if len(message.split()) == 3:
                print(message.split()[1], message.split()[0][1:], message.split()[2])
                DeviceSetter().publishCustomBind(
                    message.split()[1], message.split()[0][1:], message.split()[2][:-1]
                )
            else:
                DeviceSetter().publishCustom(
                    message.split()[1], message.split()[2], message.split()[3]
                )


async def main():
    async with websockets.serve(handler, "192.168.100.152", 8001):
        await asyncio.Future()  # run forever
```

(a) Device Setter web socket

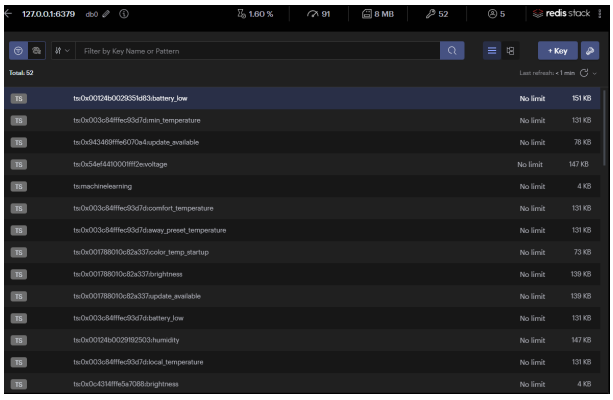
```python
CONNECTIONS = set()


async def register(websocket):
    CONNECTIONS.add(websocket)
    try:
        await websocket.wait_closed()
    finally:
        CONNECTIONS.remove(websocket)


async def send_data():
    devices = DeviceFinder()
    while True:
        await asyncio.sleep(1)
        if devices.data:
            store_data(devices.data)
            states = DeviceState(devices.data)
            websockets.broadcast(CONNECTIONS, json.dumps(states.data))


async def main():
    async with websockets.serve(register, "192.168.100.152", 5678):
        await send_data()
```

(b) Device Finder web socket



(c) Redis Database

Figure 3.12: Example of the MQTT clients incorporated into web sockets

Frontend

The frontend technology used in this case is React with Typescript. The concept of components offered by React allows for better organization of code, and better re-usability, whilst the usage of Typescript can ease the parsing of device data by using interfaces.

To allow real time data parsing and visualising, the device web sockets are opened when the application starts up, saving the device data in the state of the application using Redux. This optimizes the speed with which components that use device data are rendered, since we only need to open the web socket once, and then use the state data.

```typescript
export type DeviceStatus = Record<string, any>

export type PowerSource = "Battery" | "Mains (single phase)" | "DC Source";

export type GenericFeatureType = "numeric" | "binary" | "enum" | "text" | "list";
export type CompositeFeatureType = "fan" | "light" | "switch" | "cover" | "lock" | "composite" | "climate";

export type Endpoint = string | number;

export enum FeatureAccessMode {
    NONE,
    ACCESS_UNKNOWN = 3,
    ACCESS_STATE = 1,
    ACCESS_WRITE = 2,
    ACCESS_READ = 5,
    ACCESS_WRITE_READ = 7,
}
export interface GenericExposedFeature {
    type: GenericFeatureType;
    name: string;
    unit?: "string";
    access: FeatureAccessMode;
    endpoint?: Endpoint;
    property: string;
    description?: string;
}

export interface BinaryFeature extends GenericExposedFeature {
    type: "binary";
    value_on: unknown;
    value_off: unknown;
    value_toggle?: unknown;
}

export interface ListFeature extends GenericExposedFeature {
    type: "list";
    // bad design descision
    item_type: "number" | GenericOrCompositeFeature;
}

export interface CompositeFeature extends Omit<GenericExposedFeature, "type"> {
    type: CompositeFeatureType;
    features: GenericOrCompositeFeature[];
}
```

(a) Interfaces Snippet

```tsx
const SingleDeviceExposesContent = ({ exposes, state }: DeviceExposesProps) => {
  if (isBinaryFeature(exposes)) {
    return (
      <Grid.Col span={4}>
        <FeatureWrapperContent exposes={exposes}>
          <BinaryContent exposes={exposes} state={state} />
        </FeatureWrapperContent>
      </Grid.Col>
    );
  } else if (isNumericFeature(exposes)) {
    return (
      <Grid.Col span={4}>
        <FeatureWrapperContent exposes={exposes}>
          <NumericContent exposes={exposes} state={state} />
        </FeatureWrapperContent>
      </Grid.Col>
    );
  } else if (isEnumFeature(exposes)) {
    return (
      <Grid.Col span={4}>
        <FeatureWrapperContent exposes={exposes}>
          <EnumContent
            exposes={exposes}
            property={exposes.property}
            state={state}
          />
        </FeatureWrapperContent>
      </Grid.Col>
    );
  } else if (isLightFeature(exposes)) {
    return <LightContent exposes={exposes} state={state}></LightContent>;
  } else if (isCompositeFeature(exposes)) {
    return (
      <CompositeContent exposes={exposes} state={state}></CompositeContent>
    );
  } else if (isColorFeature(exposes)) {
    return (
      <Grid.Col span={4}>
        <FeatureWrapperContent exposes={exposes}>
          <ColorComponent exposes={exposes} state={state}></ColorComponent>
        </FeatureWrapperContent>
      </Grid.Col>
    );
  } else if (isSwitchFeature(exposes)) {
    return <SwitchContent exposes={exposes} state={state}></SwitchContent>;
  } else if (isClimateFeature(exposes)) {
    return <ClimateContent exposes={exposes} state={state} />;
  } else if (isLockFeature(exposes)) {
    return <LockContent exposes={exposes} state={state} />;
  } else {
    return null;
  }
```

(b) Rendering based on exposed type

Figure 3.13: Example of the MQTT clients incorporated into web sockets

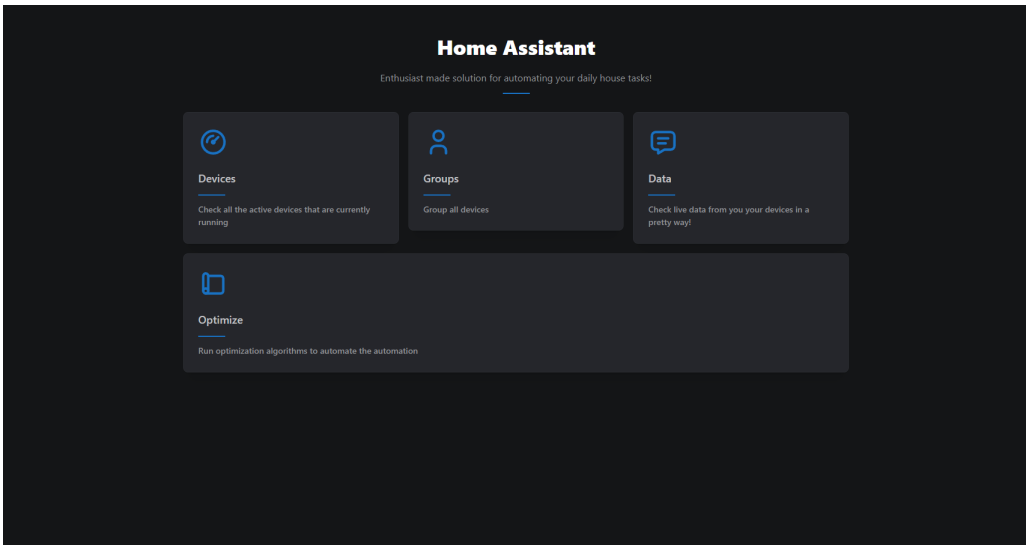
Each component then extracts the data from the state, and by using the interface system Typescript offers, we can more easily parse the device data. Example of these interfaces can be seen in figure 3.13 (a). These interfaces also serve as a way of classifying exposed data by its type. This way it is easier to create a predefined generic component for each type of exposed signal, instead of dynamically trying to figure out the type as seen in figure 3.13 (b)

The web application is split into 5 pages:

- **Home Page**
- **Devices**
- **Groups**
- **Data**
- **Optimize**

Home Page

Figure 3.14: Home Page

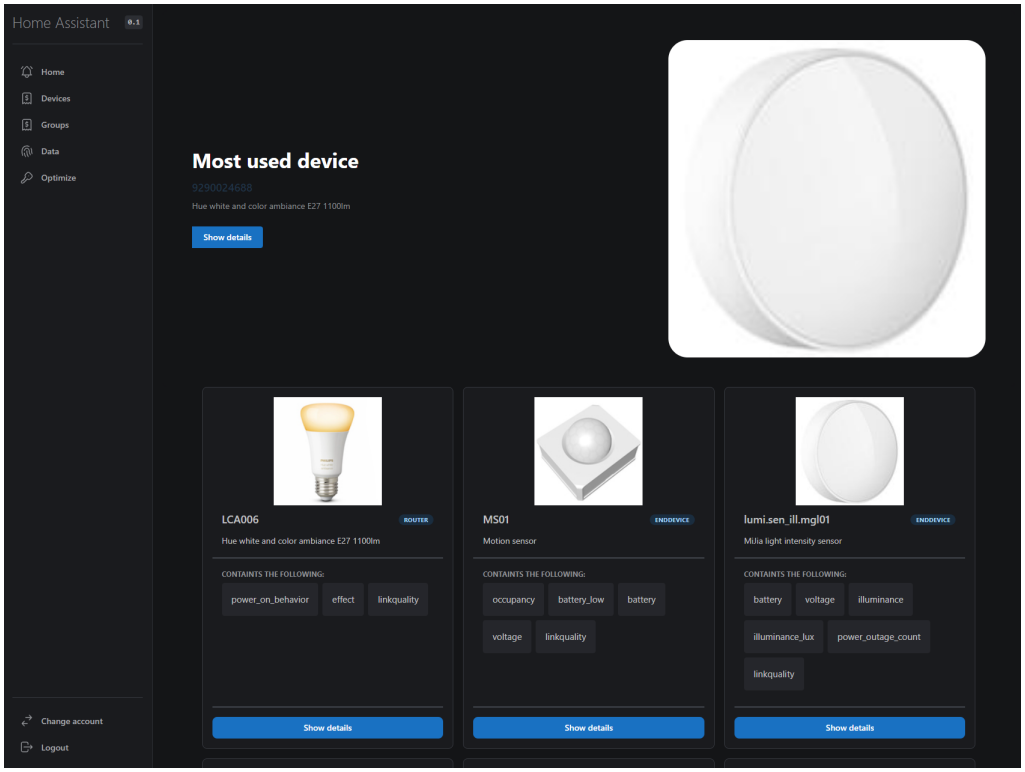


All pages adhere to the design concepts established previously. This can already be apparent from the Home Page seen in figure 3.14. Usage of grids and minimalist design allows the user to more quickly access the necessary information. By concept, this proposed solution aims to give the user only the amount of information they need, not all of it at once. Once a user reaches this screen the web sockets are started and the data collection process starts.

The usage of card like containers is also important to mention, since it will be a recurrent theme in this work.

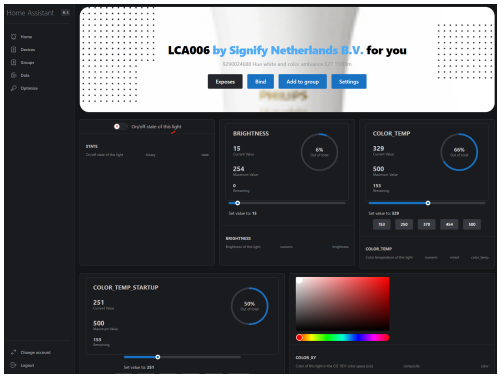
Device Page

Figure 3.15: Home Page

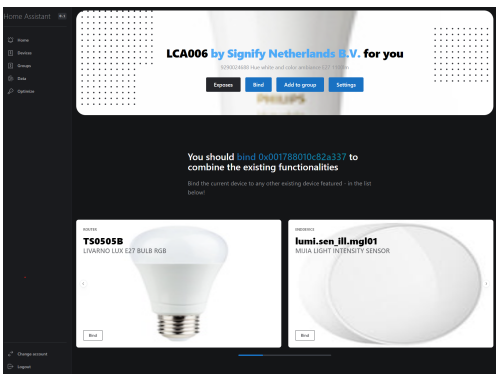


The device page uses the web sockets mentioned above to find and render all devices in the network. This can be seen in figure 3.15. Each device has a picture assigned to easier identify the device, and also some light information which can help the users decide if the device has the necessary interfaces for their needs. The introduction of a navigation bar is added from this page onward, to simplify the process of accessing different pages.

Unique Device Page



(a) Single Device exposes content



(b) Binding

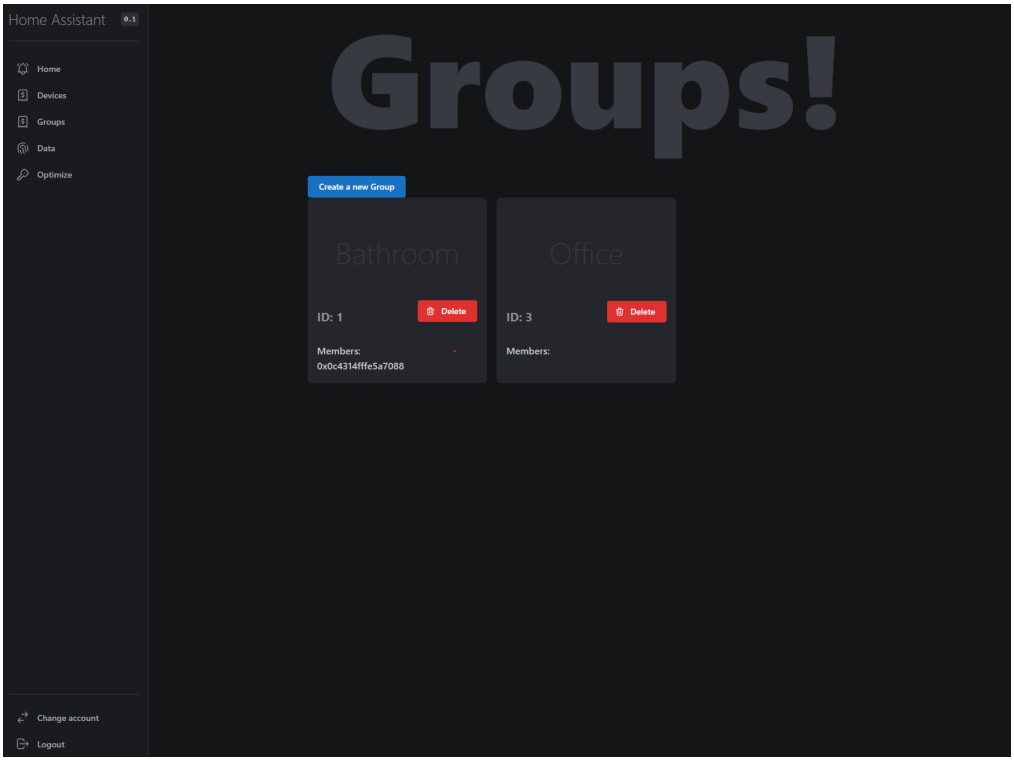
Figure 3.16: Unique device and binding

The Unique Device Page, seen in figure 3.18 (a) is a subset of Device Page. Here the user can directly interact with all the devices (which allow interactions) and be able to see the current state of the device. Here we can better see the usage of the interfaces, which are in turn used to render different type of exposed values in different manners. Numeric properties will be rendered differently from binary properties, therefore different components are created for each.

In this page binding can also be done. Using a carousel based selector, the current device selected can be binding to any item inside the carousel. If the binding fails, a notification will appear, informing the user of the failure, but also notifying if the binding was successful.

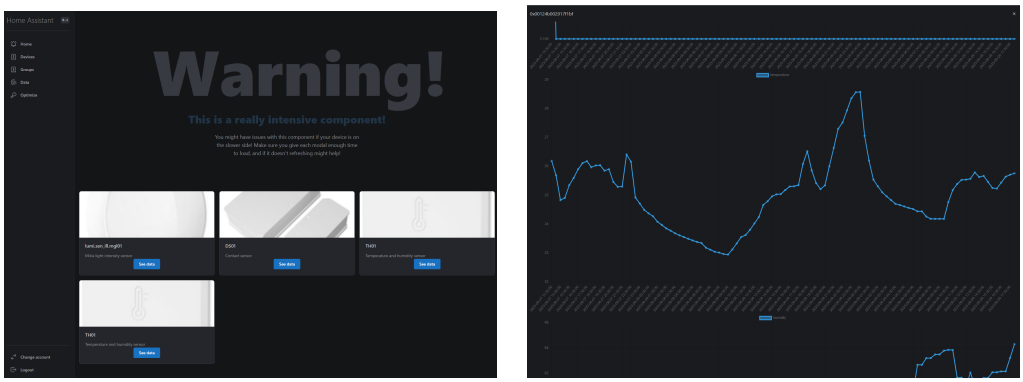
Group Page

Figure 3.17: Group Page



The group page allows the user to create and delete new groups. In the context of this work, groups don't server any purpose, so they are mostly created as proof-of-concept.

Data Page



(a) Data devices list (b) Data plotting

Figure 3.18: Unique device and binding

This page allows the user to visualize data coming from the devices. This is done with the help of the Redis database, which stores all data collected so far. That being said, it would be unrealistic to plot all of the data, considering the proposed solution collects data every second. Because of this, it is proposed to re-sample the data every hour. By doing this loading the plots is faster since we lower the amount of data.

This is important because the performance of the raspberry pi must be saved for predictions. To further optimize the page, the data is only loaded once when the component is mounted.

Optimization page

Figure 3.19: Prediction Page

This page renders the predictions realised by the machine learning model. Every time a prediction is done, it gets rendered but also pushed into the redis database. This allows the page to render a plot containing the values of past predictions.

3.2.5 Passive automation

The passive automation component for this solution is realised with the help of a machine learning model. In this section different data sets and model architectures will be discussed, with the aim of finding the best training data set and the best performing model architecture possible for the context of this work.

Three proposed data sets are used:

- Room Climate Dataset [2]
- Time Series Room Temperature Data [1]
- UCI_Data [3]

A discussion regarding each data set is required to understand which of these 3 is the better one to use for initial training.

Room Climate Data

This dataset measures room climate data, such as temperature and relative humidity. This dataset was created for the purpose of the following publication [6], which aimed to prove that room climate data ca leak privacy-sensitive information such as number of occupants.

For our purposes, we will repurpose the data for future temperature predictions. The dataset contains the following columns:

- EID: Entry ID
- AbsT: Absolute timestamp [ms]
- RelT: Relative timestamp [s]
- NID: Node ID
- Sensor Data
  - Temp: Temperature [°C]
  - RelH: Relative Humidity [
  - L1: Light Sensor 1 (Wavelength) [nm]
  - L2: Light Sensor 2 (Wavelength) [nm]
- Groundtruth
  - Occ: Number of occupants (0, 1, 2)
  - Act: Activity of occupant(s) (0 = n/a, 1 = read, 2 = stand, 3 = walk, 4 = work)
  - Door: State of Door (0 = closed, 1 = open)
  - Win: State of Window (0 = closed, 1 = open)

The data is split into multiple locations, A, B and C. Each location contains multiple files with data. To make working with this specific dataset easier, we merge all files into one single file. Besides this the absolute timestamps are converted from unix timestamps to actual dates, for easier readability.

	Temperature [C]	Relative humidity [%]
count	137361.000000	137361.000000
mean	21.551955	47.842833
std	0.696554	4.935889
min	20.420000	38.606000
25%	21.070000	43.952000
50%	21.380000	46.787000
75%	21.840000	51.971000
max	24.080000	61.246000

Table 3.1: Describing the dataset

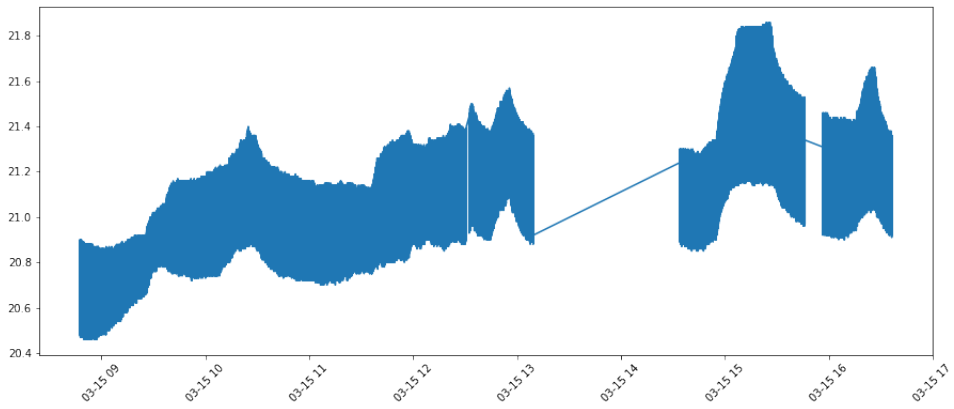
Figure 3.1 showcases the information regarding the dataset. It can be observed that there is total amount of 137361 entries in this dataset. The columns of interest for our model are the absolute timestamps, temperature and relative humidity

Absolute Timestamp [ms]	Temperature [C]	Relative humidity [%]	Date
2016-03-15 08:47:28	20.48	42.332	2016-03-15
2016-03-15 08:47:31	20.89	40.712	2016-03-15
2016-03-15 08:47:32	20.73	39.983	2016-03-15
2016-03-15 08:47:35	20.90	40.712	2016-03-15
2016-03-15 08:47:36	20.73	39.983	2016-03-15

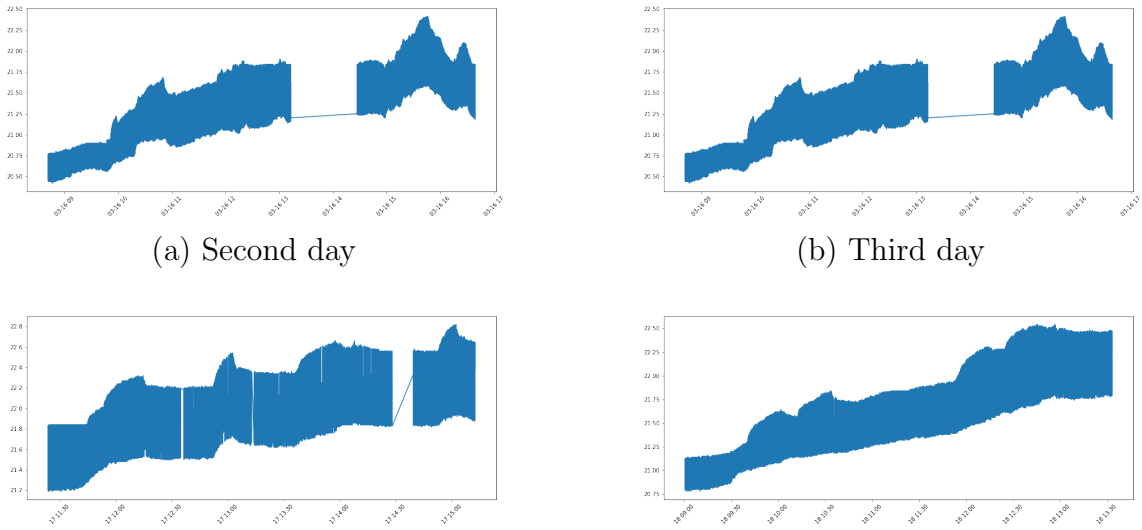
Table 3.2: First 5 rows of the dataset

From this dataset, we can then extract data based on the date, grouping it into days. The reasoning behind this, is to analyze daily patterns in temperature change before feeding this information to the model.

Figure 3.20: First day worth of temperature data



From figure 3.20, some problems can already be observed. Between the times of 13:00 and 14:30 data seems to be missing. There is no information regarding the reasoning behind this missing data, so it is assumed that either the data collection was disabled in this time frame or there was some sensor failure.



(a) Second day

(b) Third day

(c) Fourth Day

(c) Fifth Day

Figure 3.21: Next four days of data

Figure 3.27 showcases the next four days. It can be observed that the second and third day are almost identical with the first day worth of data. In continuation with the trend from the first day, there is missing data between the same time period, although the fourth day seems to shift the issue to another time frame and the fifth day resolves the issue completely. This is troublesome since the data between 13:00 and 14:30 could be extremely useful to the model. This issue can be partially solved by interpolating the missing data.

Figure 3.22: Interpolated data for the first day

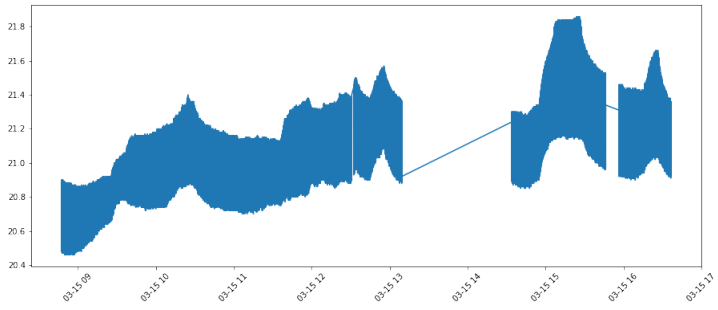


Figure 3.22 presents the new dataset after the interpolation. It is important to mention that a resampling of data was also done, so that a certain consistency could be achieved. The rule for resampling was one second and the interpolation method was time based, both offered by the pandas python library. Although the data is no longer missing, the data in the previously empty time period is linear and could affect the training of the model. Because of this adding some artificial noise to the data is proposed. To do this the most common value in change of temperature between time frames is calculated, resulting in the value of **0.34** degrees. This value is then multiplied with a random integer between -1, 0 and 1, and then added to the time frame mentioned previously.

Figure 3.23: Interpolated data for the first day with noise

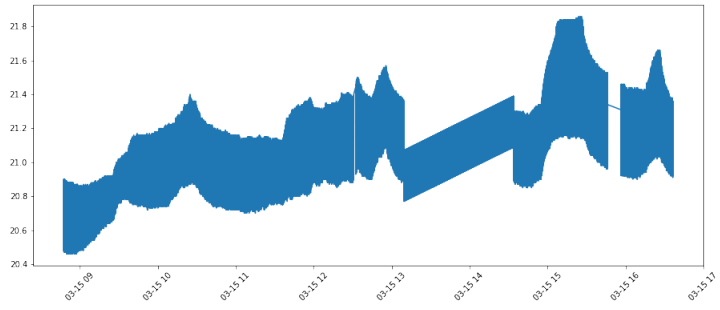


Figure 3.23 presents the results of the noise addition. Although the noise added some variety, the linearity of the data in that time frame can still be observed, and it is concluded that because of this training the model would result in bad predictions. It should be mentioned the missing time frame can be solved using other techniques, such as training the model with the missing data, and then using the model to fill in the gaps, followed by re-training it. But considering the uncertainty of this solution and the fact that multiple days are affected by this issue, this dataset was dropped in favor of other ones.

Time Series Room Temperature Data

This dataset contains 3 columns:

- Datetime1: Representing the hour of the date time column
- DAYTON_MW: Representing the value of the temperature
- Datetime: Representing the date at which the sample was taken

	Datetime1	DAYTON_MW
count	6676.000000	6676.000000
mean	11.502846	21.736007
std	6.909701	6.850502
min	0.000000	5.350000
25%	6.000000	17.512750
50%	11.000000	23.900000
75%	17.000000	26.367000
max	23.000000	36.500000

Table 3.3: Describing the dataset

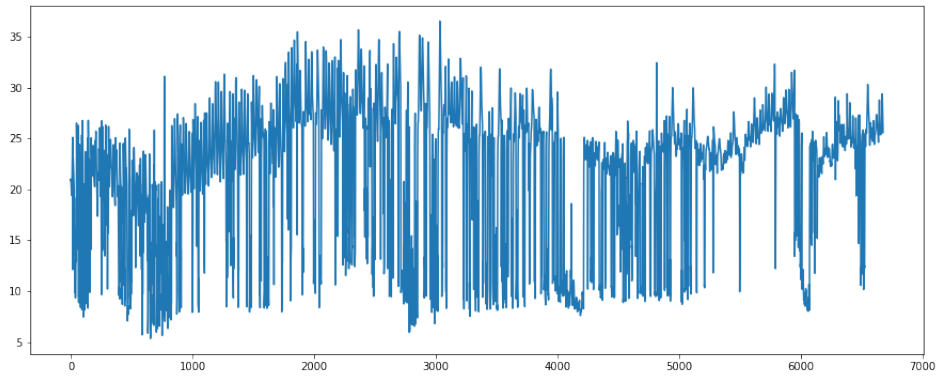
This dataset contains 6676 entries, as seen in table 3.3. This is a fairly low amount of values, and the reason for this is that the data is sampled only every hour of the day, for the duration of 278 days.

Datetime1	DAYTON_MW	Datetime
0	20.867	2022-01-04 00:00:00
1	21.000	2022-01-04 01:00:00
2	20.867	2022-01-04 02:00:00
3	20.650	2022-01-04 03:00:00
4	20.400	2022-01-04 04:00:00

Table 3.4: First 5 values of the dataset

Table 3.4 showcases the first five values of the table. It seems that the data is not out of the ordinary, and at the same time the sample rate seems to be constant.

Figure 3.24: Spikes from the dataset



That being said the rest of the data showcases some heavy spikes. Heavy drops in temperature can be seen across the dataset, and no real seasonal pattern can be observer from figure 3.24. The next step is to transform the problem to a supervised machine learning problem by generating the input and output data. For this dataset, 5 previous time steps will be used to predict the next time step. This is done by shifting the data by 5. This data is then split into test and train datasets, with a test size of 500 values out of 6676.

Figure 3.25: First LSTM model

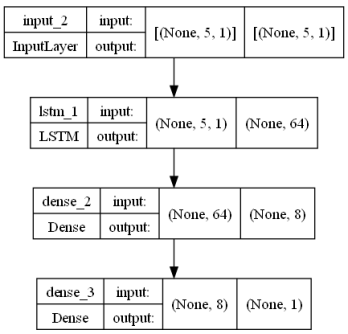


Figure 3.26 describes the model architecture. The layers are the following:

- **Input layer**. This layer defines the shape of the input data. In this case the input has a sequence length of 5 and the number of feature used for training is 1
- **LSTM layer**. LSTM layer with 64 units (memory cells)
- **Dense layer**. The first Dense layer with 8 units which helps the model learn patterns in data with the help of ReLU activation
- **Dense layer**. The last Dense layer with 1 unit which uses linear activation.

Afterwards the model is compiled with MeanSquaredError for the loss parameter, Adam optimizer with a learning rate of 0.0001, and RootMeanSquaredError metric. The model is trained for 50 epochs.

MAE	MSE	RMSE	MAPE	EVS	R2
1.8451731958618163	9.940813021850442	3.152905488886472	0.15409361659205273	0.7137427421347284	0.71

Table 3.5: Model evaluation

Figure 3.26: Model prediction plot

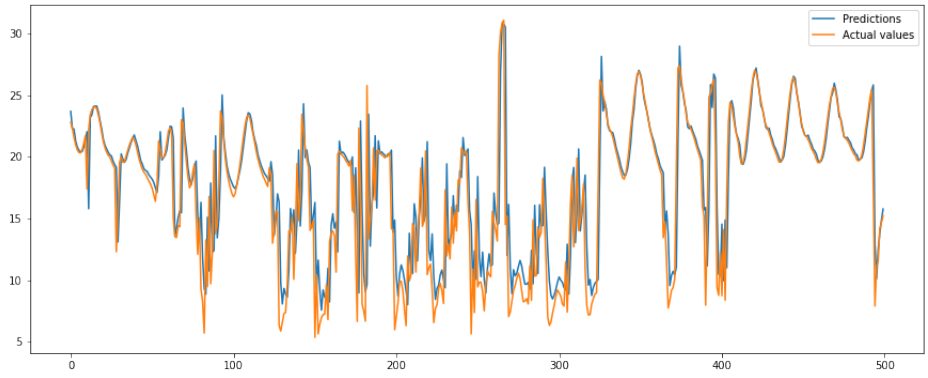
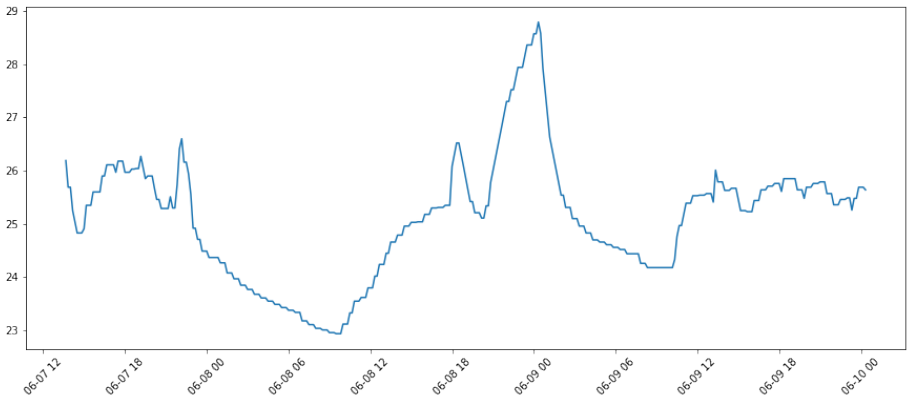
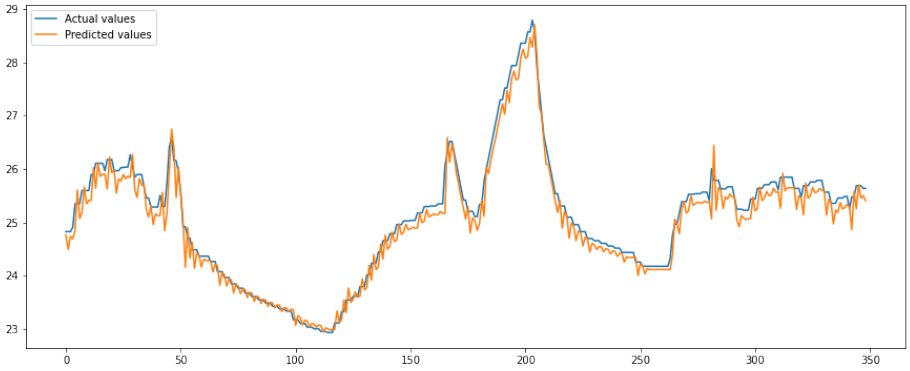


Figure 3.26 and table 3.6 showcase the model performance on the test data. The MAE and MSE are incredibly high, suggesting that model is not performing well. Despite this, the prediction are fairly accurate for the test data. This might be because the model understood certain patterns in the training data, which could cross-over to the testing data. To verify this, it is propsed to use the model against our own temperature data, from the redis database.



(a) Actual Temperature



(c) Prediction

Figure 3.27: Prediction on real data

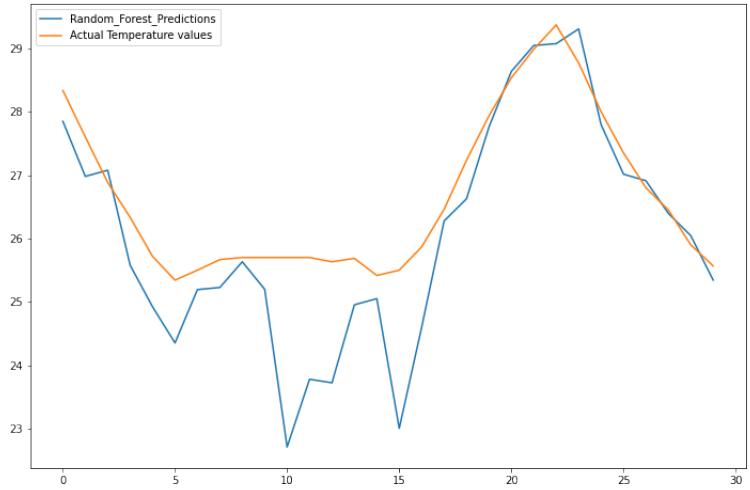
Using actual data from this work, we can see the model actually predicts in a fairly consistent matter. This, together with the light weight architecture of the model (16896 parameters) makes it a strong contender for becoming the prediction model used in the application.

For the same MLTempDataset, a Random Forest Regressor model was also trained. The number of maximum features being 5, number of estimators being 1000 and a random state of 42. The training took considerably less, but the results were lacking.

MAE	MSE	RMSE	MAPE	EVS	R2
0.6619228333333245	0.9727482397574828	0.9862800006881833	0.025386033326984966	0.5779380981352994	0.35

Table 3.6: Model evaluation

Figure 3.28: Random Forest prediction/ plot



Although from figure 3.28 the model seems to understand that there is a certain pattern to the data, the predictions for the most part are off. This is a dangerous quality of the model since if the prediction goes too low or too high, the temperature in the user's room could change dramatically and potentially consume more energy. Because of this, it is decided that the Random Forest model will not be considered as the predictor for the application.

UCI_Data

The UCI_data dataset, used for energy prediction, is a relatively large dataset with many columns, containing humidity and temperature information from 9 different rooms, as well as some external temperature and humidity information. The columns are structured as following:

- **T_[number]**, where number is the room number
- **RH_[number]** where number is the room number
- **To**, which represents the external temperature
- **RH_out**, which represents the external humidity
- **Pressure**, which represents the pressure measured in mm Hg
- **Visibility**, which represents the visibility in km

The rooms numbered from 1 to 9 each represent a unique area:

- **1.** Kitchen area
- **2.** Living room area
- **3.** Laundry room area
- **4.** Office room area
- **5.** Bathroom room area
- **6.** Outside the building (north side)
- **7.** Ironing room
- **8.** Teenager room 2
- **9.** Parents room

The dataset for each column is sampled at an interval of 10 minutes for 4.5 months. The collection process for all of this data came from a ZigBee wireless sensor network, which collected data every 3.3 minutes, but the dataset was resampled for uniformity at 10 minutes. For our purposes, we will only refer to RH_4 and T4 in this section, since it is described as an office room, which is an environment similar to the target deployment area of this model.

	date	T4	RH_4
0	2016-04-19 20:30:00	22.318571	36.610000
1	2016-03-05 04:40:00	18.700000	36.260000
2	2016-03-14 12:40:00	21.000000	34.826667
3	2016-01-22 15:30:00	16.100000	38.790000
4	2016-02-10 00:40:00	19.100000	40.900000

Table 3.7: Model evaluation

Table 3.7 showcases the first 5 values of the dataset. It is clear that the data is not sorted, since this can become troublesome later on when plotting and reading data, so it is necessary to sort it. We also have to transform the date column from strings to actual date objects. Besides this pre-processing, two new columns will be added:

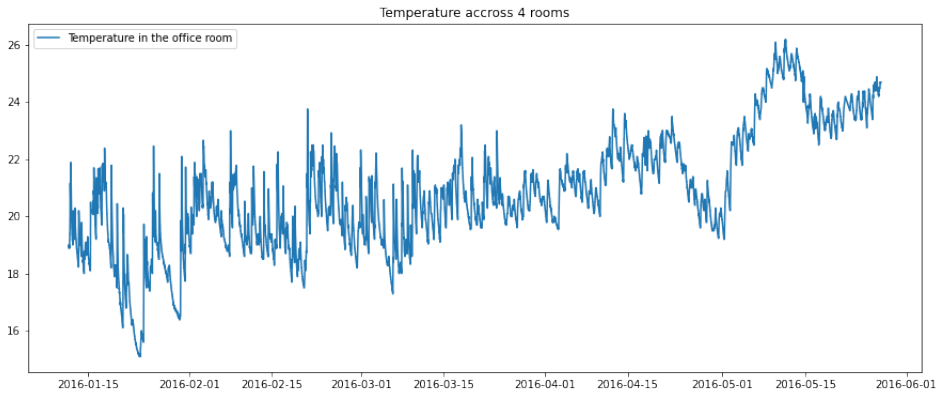
- **Day**, representing the day of the year
- **Hour**, representing the hour of day

	Day	Hour	T4
count	19735.000000	19735.000000	19735.000000
mean	79.732556	11.502002	20.855335
std	39.564943	6.921953	2.042884
min	11.000000	0.000000	15.100000
25%	45.000000	6.000000	19.530000
50%	80.000000	12.000000	20.666667
75%	114.000000	17.000000	22.100000
max	148.000000	23.000000	26.200000

Table 3.8: Model evaluation

Table 3.8 contains the description of the new modified dataset. We can see it contains 19735 total entries, which is a rather large amount of data.

Figure 3.29: Seasonal temperature changes seen in data



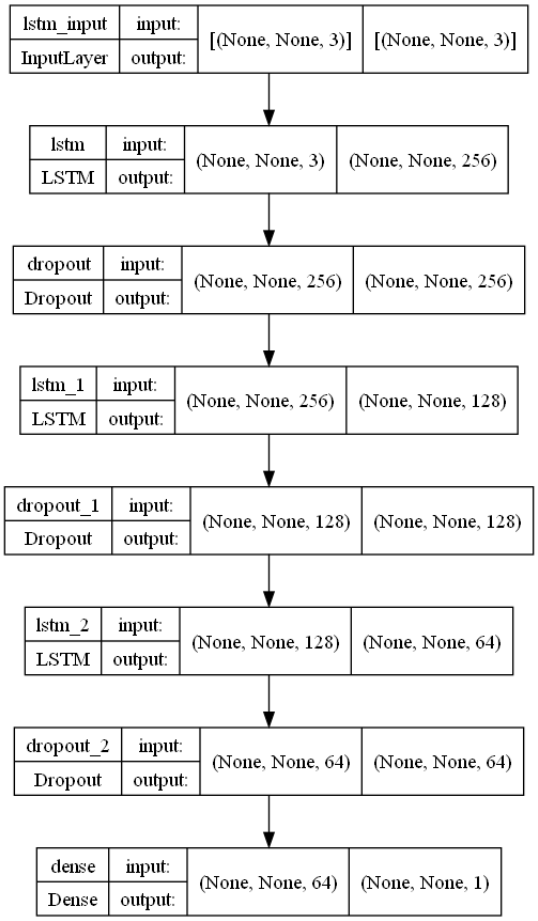
From figure 3.29 we can notice an increase in temperature with the changing of seasons. Starting from winter where we can see the lowest temperature in this dataset, up to the begging of summer where the highest temperature is recorded. This is promising since the model can use this pattern to better predict seasonal and inter-seasonal temperature changes.

The next step is to prepare the data for training. We use the same methodology used for the MLTempDataset, with some modification to create a larger, more performant model, but more computationally intensive. The first difference is the usage of scalers. Normalizing the data between 0 and 1 is a fairly common concept in preparing training data for the model. We also create a batch generator which creates random batches of data which will be used for training. This generator receives a batch size and a sequence length as parameters. The configuration chosen for this solution is :

- **batch_size** - 256
- **sequence_length** - 1440 (10 * 6 * 24), meaning we are looking 1 day into the future

This is the maximum values for each variable, anything more than this would seemingly crash the computer whilst training on a Nvidia 3080TI. The next step is to create the model architecture.

Figure 3.30: LSTM model



As it can be seen in figure 3.30, the architecture is much more complex than the previous one used in the MLTempDataset. In total there are eight layers:

- **Input layer**. This layer defines the input shape. It is important to note that this architecture allows for a variable number of time steps
- **LSTM layer**. LSTM layer with 256 units. It is also required to enable the return_sequences parameter since there are multiple LSTM layers
- **Dropout layer**. This layer is used to avoid over-fitting. 20% of the input units will be randomly set to 0
- **LSTM layer**. Second LSTM layer with 128 units, using tanh activation
- **Dropout layer**. Second dropout layer
- **LSTM layer**. Third LSTM layer with 64 units, using tanh activation
- **Dropout layer**. Third dropout layer
- **Dense layer**. Dense layer with linear activation.

A custom loss function is also created where the MSE is calculated, but a warm up parameter is added. This is done in order to ignore the begging part of the sequences, in turn allowing the model to focus on later parts of sequences, which usually are more accurate. The optimizer used for this model is the Root Mean Square Propagation optimizer with a learning rate of 0.001. Several callbacks are implemented:

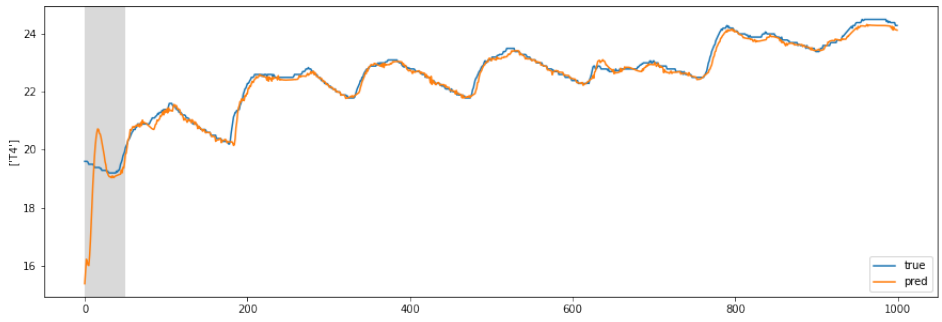
- **Checkpoint**. This is used to save the best model weights during training
- **Early Stopping**. This callback stops the training process when no more learning improvements are made. The patience of this function is 5 consecutive non-imporvements
- **Learning rate reduction**. This is used to lower the learning rate during training.

The model is trained with 50 epochs, and 100 batches per epoch. Due to the fact that the architecture is more complex, the training time increases drastically.

MAE	MSE	RMSE	MAPE	EVS	R2
0.14405093877156663	0.13723062627757687	0.37044652283099766	0.00669155930292969	0.9134770173297008	0.91

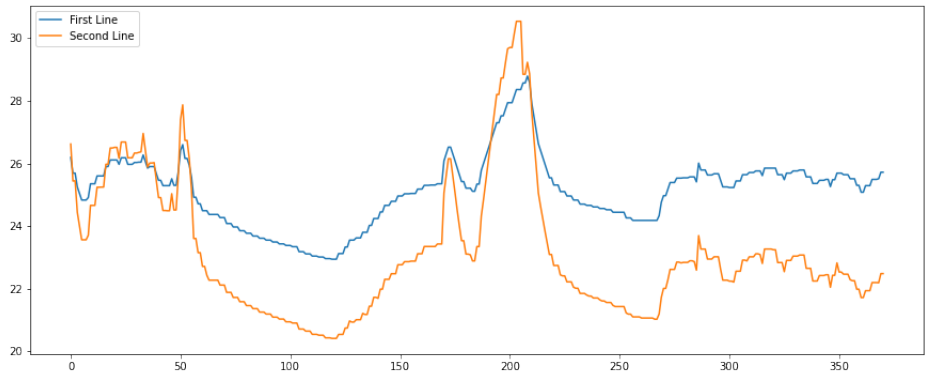
Table 3.9: Model evaluation

Figure 3.31: LSTM model2 predictions



The predictions made by the model seem to be consistent and accurate. This is reflected from the data in figure 3.31 and from the table 3.9. Further testing with the actual temperature data is needed to validate this model.

Figure 3.32: LSTM predictions for actual temperature data



It seems that the predictions are not accurate on this works dataset. From figure 3.32 we can deduce the fact that the model follows the temperature pattern, but doesn't get close to the actual values.

Deploying the model

The model trained with MLTempDataset seems to be the best performing model in regard to predicting values from our own temperature data. Because of this, this model is selected and used to predict values every 10 minutes. This is done to reduce the load on the Raspberry Pi, but also because since we are modifying the temperature of a room trough a radiator valve, a certain amount of time would be needed for the radiator to actually heat up. Therefore predictions every 10 seconds would be redundant.

Chapter 4

Conclusion

4.0.1 Active automation

From all previous chapters, certain results can be observed. The active automation component of this proposed solution can be considered a success, although from an arguably biased source (friends and family). From an objective point of view, the usage of modern web design pattern such as grids, cards, responsive components, real time data visualisation and minimal design, a user should have an easier time navigating the application, and interacting with the devices.

4.0.2 Passive automation

In total 3 models have been trained on different datasets:

Model	MAE	MSE	RMSE	MAPE	EVS	R2
UCI_Data LSTM	0.14405093877156663	0.13723062627757687	0.37044652283099766	0.00669155930292969	0.9134770173297008	0.91
MLTempDataset Random Forest	0.6619228333333245	0.9727482397574828	0.9862800006881833	0.025386033326984966	0.5779380981352994	0.35
MLTempDataset LSTM	1.8451731958618163	9.940813021850442	3.152905488886472	0.15409361659205273	0.7137427421347284	0.71

Table 4.1: Model evaluation

The results collected from the different models can be considered successful as seen in table 4.1, although further research regarding different architectures and different datasets is required. Further optimizations can also be done, reducing the computing power needed for predictions, therefore increasing the overall speed of the project as a whole, since it is running on limited hardware.

4.0.3 Overall

From an overall point of view, the proposed solution manages to demonstrate that combining passive and active automation is possible in the context of IoT home automation projects. That being said, the practicality of such a project is debatable when considering the multitude of possible sensor configurations, meaning that a new sensor containing the temperature in Kelvin instead of degrees Celsius, could generate issues with both the passive automation and the active automation. These corner cases can be avoided with further research, and development on the application.

Bibliography

[1] MLTempDataset, howpublished = https://www.kaggle.com/datasets/vitthalmadane/ts-temp-1,.

[2] RoomClimateDatasets, howpublished = https://github.com/iotsec/room-climate-datasets,.

[3] UCIDataset, howpublished = https://www.kaggle.com/datasets/subh86/uci-data,.

[4] Zigbee2MQTT, howpublished = https://www.zigbee2mqtt.io/,.

[5] Zigbee2MQTTInstall, howpublished = https://www.zigbee2mqtt.io/guide/installation/01_linux.html,.

[6] Frederik Armknecht, Zinaida Benenson, Philipp Morgner, Christian Müller, and Christian Riess. Privacy implications of room climate data. *Journal of Computer Security*, 27(1):113–136, 2019.

[7] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[8] Temesegan Walelign Ayele and Rutvik Mehta. Real time temperature prediction using iot. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pages 1114–1117, 2018.

[9] Tushar Chaurasia and Prashant Kumar Jain. Enhanced smart home automation system based on internet of things. In *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 709–713, 2019.

[10] Pooja S. Chinchansure and Charudatta V. Kulkarni. Home automation system based on fpga and gsm. In *2014 International Conference on Computer Communication and Informatics*, pages 1–5, 2014.

[11] Vehbi C. Gungor and Gerhard P. Hancke. Industrial wireless sensor networks: Challenges, design principles, and technical approaches. *IEEE Transactions on Industrial Electronics*, 56(10):4258–4265, 2009.

[12] Pradeep Hewage, Marcello Trovati, Ella Pereira, and Ardhendu Behera. Deep learning-based effective fine-grained weather forecasting model. *Pattern Analysis and Applications*, 24(1):343–366, 2021.

[13] Md. Sajjad Hossain Shawon, Chinmoy Das, Md. Tabil Ahammed, Gobinda Biswas, MD. Shaurov Mia, Emima Akter Eva, and Md. Nazmus Sakib. Voice controlled smart home automation system using bluetooth technology. In *2021 4th International Conference on Recent Trends in Computer Science and Technology (ICRTCST)*, pages 67–72, 2022.

[14] Sean Dieter Tebje Kelly, Nagender Kumar Suryadevara, and Subhas Chandra Mukhopadhyay. Towards the implementation of iot for environmental condition monitoring in homes. *IEEE Sensors Journal*, 13(10):3846–3853, 2013.

[15] Jin-Shyan Lee, Yu-Wei Su, and Chung-Chou Shen. A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi. In *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*, pages 46–51, 2007.

[16] M. Mahith, Darshan S.B. Kumar, K.C. Prajwal, and M. Dakshayini. Bluetooth home automation. In *2018 Second International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 603–607, 2018.

[17] V.L.K. Bharadwaj Manda, Voona Kushal, and N. Ramasubramanian. An elegant home automation system using gsm and arm-based architecture. *IEEE Potentials*, 37(5):43–48, 2018.

[18] M. Victoria Moreno, Antonio F. Skarmeta, Alberto Venturi, Mischa Schmidt, and Anett Schuelke. Context sensitive indoor temperature forecast for energy efficient operation of smart buildings. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 705–710, 2015.

[19] Neaz Md. Morshed, G.M. Muid-Ur-Rahman, Md. Rezaul Karim, and Hasan U. Zaman. Microcontroller based home automation system using bluetooth, gsm, wi-fi and dtmf. In *2015 International Conference on Advances in Electrical Engineering (ICAEE)*, pages 101–104, 2015.

[20] Debayan Paul, Tanmay Chakraborty, Soumya Kanti Datta, and Debolina Paul. Iot and machine learning based prediction of smart building indoor temperature. In *2018 4th International Conference on Computer and Information Sciences (ICCOINS)*, pages 1–6, 2018.

[21] D. Pavithra and Ranjith Balakrishnan. Iot based monitoring and control system for home automation. In *2015 Global Conference on Communication Technologies (GCCT)*, pages 169–173, 2015.

[22] T M Rubaith Bashar Sezan, Munem Shahriar, Mohammad Munem, and Tasnim Binte Shawkat. Cost effective bluetooth technology based home automation system using smart-phone application. In *2019 3rd International Conference on Electrical, Computer & Telecommunication Engineering (ICECTE)*, pages 284–287, 2019.

[23] Harsh Kumar Singh, Saurabh Verma, Shashank Pal, and Kavita Pandey. A step towards home automation using iot. In *2019 Twelfth International Conference on Contemporary Computing (IC3)*, pages 1–5, 2019.

[24] Rozita Teymourzadeh, Salah Addin Ahmed, Kok Wai Chan, and Mok Vee Hoong. Smart gsm based home automation system. In *2013 IEEE Conference on Systems, Process & Control (ICSPC)*, pages 306–309, 2013.

[25] James E Tomkayo. Anecdotes. *IEEE Annals of the History of Computing*, 16(03):59–61, 1994.

[26] Vishakha D. Vaidya and Pinki Vishwakarma. A comparative analysis on smart home system to control, monitor and secure home, based on technologies like gsm, iot, bluetooth and pic microcontroller with zigbee modulation. In *2018 International Conference on Smart City and Emerging Technology (ICSCET)*, pages 1–4, 2018.

[27] Alex van der Zeeuw, Alexander JAM van Deursen, and Giedo Jansen. The orchestrated digital inequalities of the iot: How vendor lock-in hinders and playfulness creates iot benefits in every life. *New Media & Society*, 0(0):14614448221138075, 0.

[28] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Jubert, Margaretha Mazura, Mark Harrison, Markus Eisenhauer, and Pat Doody. Internet of things strategic research roadmap. 01 2009.

[29] Jingchao Yang, Manzhu Yu, Qian Liu, Yun Li, Daniel Q. Duffy, and Chaowei Yang. A high spatiotemporal resolution framework for urban temperature prediction using iot data. *Computers & Geosciences*, 159:104991, 2022.

[30] Manzhu Yu, Fangcao Xu, Weiming Hu, Jian Sun, and Guido Cervone. Using long short-term memory (lstm) and internet of things (iot) for localized surface temperature forecasting in an urban environment. *IEEE Access*, 9:137406–137418, 2021.