Masaryk University
Faculty of Informatics

Node-RED Smart Home Flows

Bachelor's Thesis

Lukáš Hasík

Brno, Spring 2021

Masaryk University
Faculty of Informatics

Node-RED Smart Home Flows

Bachelor's Thesis

Lukáš Hasík

Brno, Spring 2021

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Lukáš Hasík

Advisor: Ing. Lukáš Grolig

Acknowledgements

I would like to thank my thesis advisor Ing. Lukáš Grolig for his immense patience, advice, and support.

Keywords

Flow-based programming, Node-RED, smart home, module

Table of contents

1. Introduction 1
2. Commercial Solutions 1
   2.1. Amazon.com 2
      2.1.1. Introduction 2
      2.1.2. Amazon Alexa 2
      2.1.3. Available Options 2
      2.1.4. Compatibility 3
      2.1.5. Conclusion 3
   2.2. Lidl Home 3
      2.2.1. Introduction 3
      2.2.2. Available Options 3
      2.2.3. Conclusion 4
   2.3. Loxone 4
      2.3.1. Introduction 4
      2.3.2. Available Options 4
      2.3.3. Conclusion 5
   2.4. Summary 5
3. Common modules 6
   3.1. Lights 6
   3.2. Temperature control 6
   3.3. Security 6
   3.4. Shading 7
4. Node-RED 8
5. Implementation of the Common Flows 9
   5.1. Inject Node 9
   5.2. Debug Node 9
   5.3. Switch Node 9
   5.4. Function Node 10
   5.5. Link Nodes 10

5.6. Delay Node 10
6. Modules 11
6.1. Light module 11
6.1.1. Introduction 11
6.1.2. Basic light flow 11
6.1.3. Motion activated light flow 12
6.1.4. Patio light flow 14
6.1.5. Summary 15
6.2. Alarm module 16
6.2.1. Introduction 16
6.2.2. Implementation 16
6.2.3. Summary 17
6.3. Temperature control module 18
6.3.1. Introduction 18
6.3.2. Implementation 18
6.3.3. Summary 20
6.4. Shading module 20
6.4.1. Introduction 20
6.4.2. Advanced shading flow 20
6.4.3. Simplified blinds flow 22
6.4.4. Summary 23
6.5. Smart home implementation 23
7. Final conclusion 25
8. Bibliography 26

Table of Figures

Figure 6.1.2. Basic light flow implementation 11
Figure 6.1.3. Motion activated light flow implementat 13
Figure 6.1.4. Patio light flow implementation 14
Figure 6.2. Alarm flow implemtation 16
Figure 6.3. Temperature control implementation 18
Figure 6.4.2. Advanced shading flow implementation 21
Figure 6.4.3. Simplified shading flow implementation 23
Figure 6.5. Sample of smart home inputs 25

1. Introduction

Smart homes are defined as houses designed to maximize the comfort of their inhabitants by means of automation of their devices. Devices can be automated or controlled remotely via phones and tablets. Smart homes can monitor and control attributes of a house, such as lighting, alarm, heating, and many others. Devices in smart homes can not only be connected to each other but also to the internet. Such devices are part of the *Internet of things*, a network of objects exchanging data via the internet. [1] [2]

However, the world of smart homes is not yet big enough to be standardized and available to everyone. That is partly due to the lack of awareness about smart homes, which are still perceived more as a thing of the future than a thing of the present. [3] Another factor is the lack of preset flows, that could be easily downloaded, edited, and used by anyone interested in upgrading their home to a smart home. Searching the internet for such flows yield very little useful results.

This thesis focuses on the purchasable options of preset smart homes available on the market in the first chapter. The second chapter describes most common modules used in most smart homes. The third chapter focuses on the usage of Node-RED, flow-based development tool used to implement the preset flows of the most common modules. The fourth chapter lists the most common nodes from Node-RED used in the implementation to ease the reader into chapters five to eight, which are focused on the description of the flows and the methods used in the implementation. The ninth chapter features a model smart home implemented by using previously designed preset flows.

2. Commercial Solutions

This chapter describes three different options of smart homes available on the market. There are, of course, many more options available, but generally, two types of smart home implementation can be found. The first type is the implementation via purchasing a central unit, for example Amazon Alexa, and other devices capable of a wireless connection to the central device. The central unit is not necessary, and devices can be

controlled by using a phone or a tablet. The other type is implemented by the company itself, custom-built to fit the house.

Three companies were chosen. Amazon with Amazon Alexa as one of the most common custom-built smart home options, Lidl Smart Home, as a surprising but inexpensive and simple option. Finally, the Loxone company offering bespoke smart home modules.

This chapter's purpose is not intended to promote or demote a particular company. Its content is purely descriptive.

2.1. Amazon.com

2.1.1. Introduction

American company Amazon.com offers a wide variety of smart home devices. The fundamental component of smart homes with Amazon.com devices is the *Amazon Echo*; a voice-activated, internet-connected smart speaker using *Amazon Alexa* to carry out voice commands. *Amazon Echo* and *Alexa* are terms often used mistakenly interchangeably. [4] To prevent further confusion, let it be known that *Amazon Echo* is the smart speaker device, and *Amazon Alexa* is the cloud-based service, which can understand voice commands used by *Echo* to carry the commands out. *Amazon Echo* is not needed for a smart home to function properly and can be substituted by using the *Amazon Alexa* phone app. This however requires opening the application and clicking the central *Talk* button, and as such is not as comfortable to use as the *Amazon Echo*. [4]

2.1.2. Amazon Alexa

As was mentioned before, *Amazon Alexa* is a cloud-based service, able to recognize voice commands. *Alexa* can not only control other devices; its connection to the internet also means *Alexa* can use the internet and other applications, which can be added, or find answers to questions asked. [4]

2.1.3. Available Options

Virtually any home appliance or device imaginable with *Alexa Compatibility* on the Amazon.com webstore[1]. Ranging from smart plugs to televisions, kitchen appliances, locks, cameras and even pet toys, with *Amazon Alexa*, any device can be automated and controlled remotely or using voice commands. [5] To list some of the available possibilities, Alexa can operate the lights and the windows blinds to turn

[1] www.amazon.com

on and open at a certain time as a form of an alarm clock. Alexa can also lock or unlock doors with compatible locks installed and turn on other appliances, such as TV or sound systems.

2.1.4. Compatibility

One of the most significant advantages of *Amazon Alexa* is the compatibility with other devices. *Alexa* is no obscure device and is very well known in the current world amongst technology enthusiasts, and as such, there is no shortage of devices made by various companies which are compatible with *Amazon Alexa*. [5] The information about products compatibility with *Amazon Alexa* can be found in products description. Sometimes, a bridge device is needed to connect a device to *Amazon Alexa*.

2.1.5. Conclusion

In 2019, *Amazon Alexa* home assistant emerged as the most used home assistant on the market. This fact can be attributed to the success of *Amazon Echo* smart speaker. [6] *Amazon Alexa* is a prevalent choice amongst technology enthusiasts, partly due to a large number of products compatible with *Amazon Alexa*. It is also effortless to install, add devices and create groups of devices. *Amazon Alexa* being voice activated is also a reason why is it so popular, with voice activation currently being the most popular method of control [7]. However, those interested in more bespoke modules may find it lacking in customizability.

2.2. Lidl Home

2.2.1. Introduction

The Lidl Home is a name for a collection of devices by the German retail chain Lidl. Currently, the Lidl Home is focused mainly on the implementation of lights. The budget-friendly set features numerous types of light bulbs, with RGB light bulbs included, smart plugs and a gateway device, used to connect other devices. The communication between devices is handled by the *Zigbee 3.0* standard. The *Zigbee 3.0* standard is functionally similar to the Bluetooth standard. [8]

2.2.2. Available Options

As was mentioned before, the Lidl Home focuses mainly on the control of lights. Other simple devices can be also controlled using smart plugs. Using the smart plugs, devices can be remotely turned on or off. Smart plugs offer no advanced functionality beyond turning devices on and off. [8]

Lidl Home features a wide variety of light bulbs, some with RGB support. All devices can be added to the home network using the Lidl Home application, available both on iOS and Android OS. [8]

2.2.3. Conclusion

While the Lidl Home does not offer much in terms of smart home modules, its implementation of the light's module is very well done, easy to use, and budget-friendly. It is an ideal choice for smart home beginners as it is easy to install and control. However, those interested in more extensive home automation might find it lacking in functionality.

2.3. Loxone

2.3.1. Introduction

Loxone company, founded in 2009, based in Kollerschlag – Austria, is a company focusing on the design and implementation of building automation. Loxone specializes in implementing smart residential houses and also smart offices, hotels, and restaurants. [9]

2.3.2. Available Options

Unlike previously mentioned companies, Loxone does not focus on smart home components, but rather on the entire implementation. Loxone company offers a wide range of available smart home modules:

1. Lights
2. Shading
3. Temperature control
4. Energy management
5. Audio
6. Security
7. Wellness

Individual modules are capable of cooperation with other modules. [8] These components are controlled via an installed mini server, serving as a central hub. Central hubs are not definite and can be further expanded by other Loxone modules. The module installation is not an easy process and should be handled by designated personnel. Loxone modules can be controlled via phone or a tablet device in a designated application and are also installed with a custom control panel specifically designed to operate the specific module. [9]

2.3.3. Conclusion

Loxone company offers a wide variety of modules. Modules are very well designed, having complex functionality and useful features. Modules can be added later to the central hub, and as such, upgrades and updates are easy to implement. However, the installation itself should always be handled by professionally trained personnel and therefore makes a customer dependent on the company. This, along with the fact that the customer cannot alter the modules, could become a significant downside.

2.4. Summary

This chapter explored available smart home options and listed some options with a more in-depth description. These options, however, do not represent the entire smart home market. There are many more options, but most of the options are similar to at least one of the previously mentioned companies. Amazon's *Alexa* can be rivaled by Apple's HomePod using Siri or the Google Assistant [6], and other companies, offering custom-built home automation, similar to Loxone exist.

3. Common modules

Upon examination of the available market choices in the previous chapter, it became clear that the most common modules are:

1. Lights
2. Temperature control
3. Security
4. Shading

Other less common modules were, for example, an automatic water sprinkler or an automated sound system.

3.1. Lights

Lights are a necessity in every house and, as such, are the most common modules and offer a wide variety of additional functionality. Not only can lights be controlled remotely, but their brightness or color can also be manipulated. Lights can now be connected to an alarm system and turn on whenever an alarm is triggered or can simulate a presence in a home while its inhabitants are not present and deter possible intruders. With these features and many others, it becomes clear that lights are a vital part of every smart home.

3.2. Temperature control

Temperature control is vital to keep the house in good condition. Smart, automated temperature control is designed to not only maintain the optimal temperature in a house to preserve it but also to run as economically effectively as possible but also ecologically. That can be ensured by a detector of presence, meaning the room temperature will be kept lower if no presence is detected. Similar to lights, the temperature in a house can be altered and viewed remotely via a phone or a tablet. Smart temperature control knows when to turn on the heating and when to decrease the temperature by using the air conditioning.

3.3. Security

One of the key modules, security is vital for house inhabitants to feel safe and secured. Safety can be assured by multiple possible devices, such as alarms activated by various sensors, electronic or password-protected locks, and security cameras. Security modules can also interact with other house modules, such as lighting or blinds, to alert the house inhabitants of danger and helping them escape.

3.4. Shading

At first glance, a simple module, the shading module can field a wide variety of features. The more obvious ones focus on the essential operation of blinds or shades, such as opening or raising. More advanced shading modules can also be automatically operated and open in the morning or lower the blinds if they detect sunlight. Advanced shading modules can interact with other modules, such as the alarm module, by raising the blinds so the room inhabitants can escape, or the temperature control module, keeping the room cool by blocking the sunlight.

4. Node-RED

Node-RED is the primary technology used in this thesis. That is due to the fact that Node-RED is open-source and very easy to learn and operate and as such it is the perfect tool to implement smart home flows. This chapter focuses on the introduction of Node-RED and explaining its functionality and describing some of its features.

Node-RED is a programming tool, developed in 2013 by Nick O'Leary and Dave Conway-Jones, used for connecting hardware, APIs, and devices together. [10] It can be installed either locally, on a device like Raspberry Pi, or in the cloud and uses a browser-based flow editor. Using a wide range of nodes, further expandable by importable libraries, Node-RED allows for an easy way to build, connect, and deploy various flows. With a lightweight runtime built on Node.js, JavaScript functions can be created within nodes. [1] The design and functionality of nodes in Node-RED are intuitive and well documented on the official website. The easy-to-understand flow-based programming allows even for an inexperienced user to grasp basic concepts quickly, and therefore, Node-RED is the perfect tool for anyone with basic programming skills to configure their smart home to their liking.

Node-RED, apart from nodes, offers a dashboard, a panel that serves as a graphical interface and can be used to further improve and customize flows. This feature would prove to be confusing to a new user and therefore is not suited to fulfill the goal of this thesis. However, should the topic of this thesis be expanded, in the future, by implementing smart home modules along with a graphical interface to operate said modules, the dashboard feature would provide the functionality to do so.

5. Implementation of the Common Flows

To keep flows unified and homogenous as much as possible, a process about how to create flows had to be devised. While Node-RED contains libraries with very powerful nodes, to keep the implementation simple, only the core nodes were used as their functionality is very easy to understand, and as such, they are the most suited for the task of creating simple preset flows.

Before the description of individual flows, the most used nodes and practices should be introduced. To store values outside or inside nodes, there are three options available. The value can be stored either in the context object (which stores a value for a node), the flow object, (which stores a value for a flow) and the global object (storing the value for the entire project). This implementation uses the flow object for storage solely. With the data storage possibilities of Node-RED described, various used nodes should be described.

5.1. Inject Node

The inject node is the basic input node present in every flow, sometimes even multiple times. Upon clicking on the inject node, a message JSON object is sent to the flow. This message object can contain various values in its attributes. These values can, but do not have to, be used in certain flows. For example, in flows operating lights, the message contains a payload attribute carrying a number to indicated which light is being used. In other cases, the message does not have to carry any additional values nor have any attributes. The attributes can be accessed by other nodes in the flow connected to the inject node and the values can be manipulated, this is done very frequently in this implementation.

5.2. Debug Node

The debug node is used as a basic output node and as a descriptive node informing the user about the status of the flow in certain flows. It serves to simulate an actual device that would be operated by the flow, a light, for example. The debug node can be set to print either only certain attributes of the message it received or the entire message object, for example. In this implementation, the message payload attribute is printed.

5.3. Switch Node

Another frequently used node is the switch node. It is used to evaluate a Boolean expression and determine which way the flow should continue. It can compare values stored either in the flow object, the message object, the global object, or an

environmental variable. It can also compare values from mentioned objects to numbers and strings.

5.4. Function Node

The most powerful node and most frequently used node is the function node. While initially empty, a JavaScript code can be written inside them, and they can access data from the flow or the message and change them without restriction. Function nodes are also used to substitute Change nodes, nodes used to change data stored in the flow or message, because of their more advanced functionality and the ability to deal with yet uninitialized values. Function nodes are the backbone of every flow and are very easy to implement.

5.5. Link Nodes

Link nodes constitute of Link-in and Link-out nodes, both of which are used to connect flows via virtual wire. The Link-out node serves as an output node, sending a wire to a different flow, while the Link-in node serves as an input, catching wires sent from other flows. These nodes are vital and offer interconnectivity of flows and are used, for example, in the light module, where the motion-activated light provides only the functionality of the timer and motion sensor while the actual light functionality is handled in the basic light flow. They are also used in the smart home implementation flow to connect actual switches to the preset flow. Link nodes have a Many-to-Many relationship, meaning that a single Link-out can lead to multiple flows while a single Link-in node can catch signals from more than one flow.

5.6. Delay Node

More an auxiliary node than an actual functional node, the delay node halts the flow for a set amount of time. This is done to simulate an operation of an actual device. While the delay node has some purpose in certain flows, in the implemented preset flows, except the motion activated light flow, they serve only to mimic real behavior.

6. Modules

This chapter describes the implementation and the process of the flow from start to finish. Figures are provided to each flow to aid the reader into the topic.

6.1. Light module

6.1.1. Introduction

One of the most common modules is the light module. Light flows variety can range from simple lights controlled by a manual switch, to complex remotely controlled lights with various sensors, such as motion sensor or a daylight sensor. Lights can not only be turned on or off, but their brightness can also be changed and sometimes even the light's hue. To fit the thesis theme, only some light flows have been implemented. Basic light flow, motion activated light flow and time activated light flow have been selected.

6.1.2. Basic light flow

This flow represents a basic light which can be either turned on or off and whose brightness can be changed to values ranging from fifty to one hundred by the increments of ten. Before manipulating the flow, a sub-flow designed to initiate all lights has to be run. Upon triggering this sub-flow via the *Init./Reset lights* node, an array filled with objects storing attributes such as brightness or status is created. Initially the brightness attribute is set to the maximum of one hundred and the status is set to *Off*. When the array is initialized, the debug node prints a message informing us that all is well. Basic light flow can now be safely run.

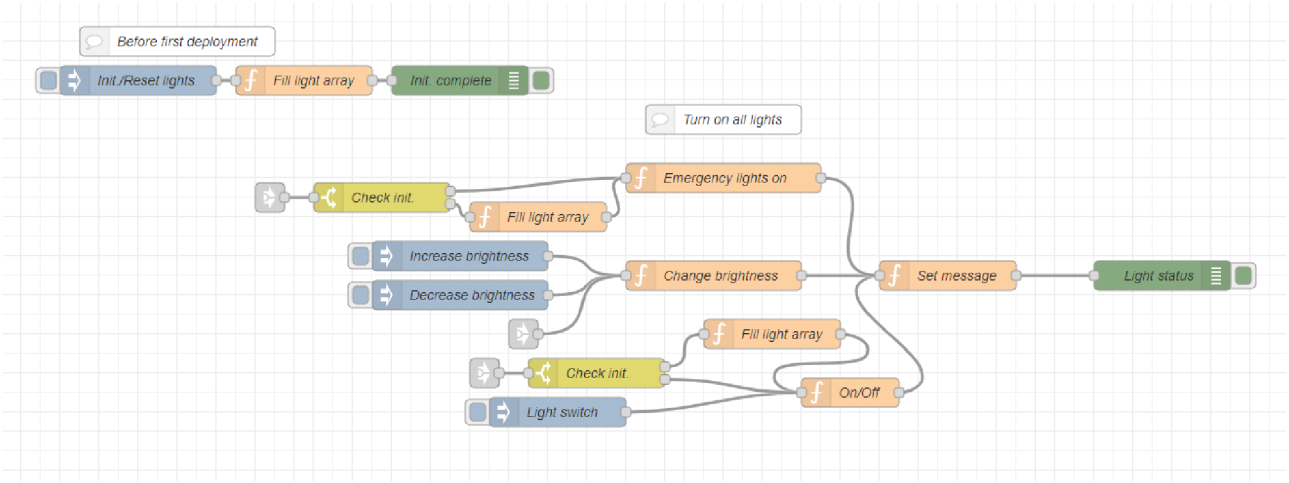

Figure 6.1.2. Basic light flow implementation

First input of this flow is the *Light switch* input node. This node sends out a message whose payload contains a number. This number represents a position in the previously created array of objects. When the flow reaches the *On/Off* node, the code accesses the object in the array on the position of the message payload number and simply reverses the status. Then the flow continues to the *Set message* function node, which collects all the desired information and connects them into one message which is then printed by the *Light status* debug node, here representing the actual light.

Other inputs are the *Increase brightness* and *Decrease brightness* input nodes. These nodes like the light switch node carry a number in the message payload which is then used in the *Change brightness* function node. There the light object is obtained same as before using the payload as an index in the array and its brightness attribute is either increased or decreased by the increment. The light status must be *On*. Afterwards, the flow once again continues to the *Set message* function node and then prints the message in the debug node. There are also three link-in nodes. One to connect all light switches, second one to connect all lights' brightness control and finally the link-in node used in the case of alarm. This last link-in node leads to a switch node where the initialization status of the light array is checked and initializes it in case it has not been done before. Then the flow follows to the *Emergency lights on* function node which simply turns all lights on and sets their brightness to the maximal value of one hundred.

This flow could benefit from the usage of more advanced nodes that Node-RED has to offer, such as the slider or color picker, however these nodes function only when a real connectable device is detected and as such cannot be used to fit requirements of this thesis.

6.1.3. Motion activated light flow

The other very common variation of a light flow is the motion activated light flow. It is essentially a variation of the basic light flow with the added motion sensor and manual control sub-flow. The flow is connected to the basic light flow by the link out nodes so instead of manipulating the light itself, this flow only sets when and for how long the light should be turned on and the rest is handled by the basic light flow.

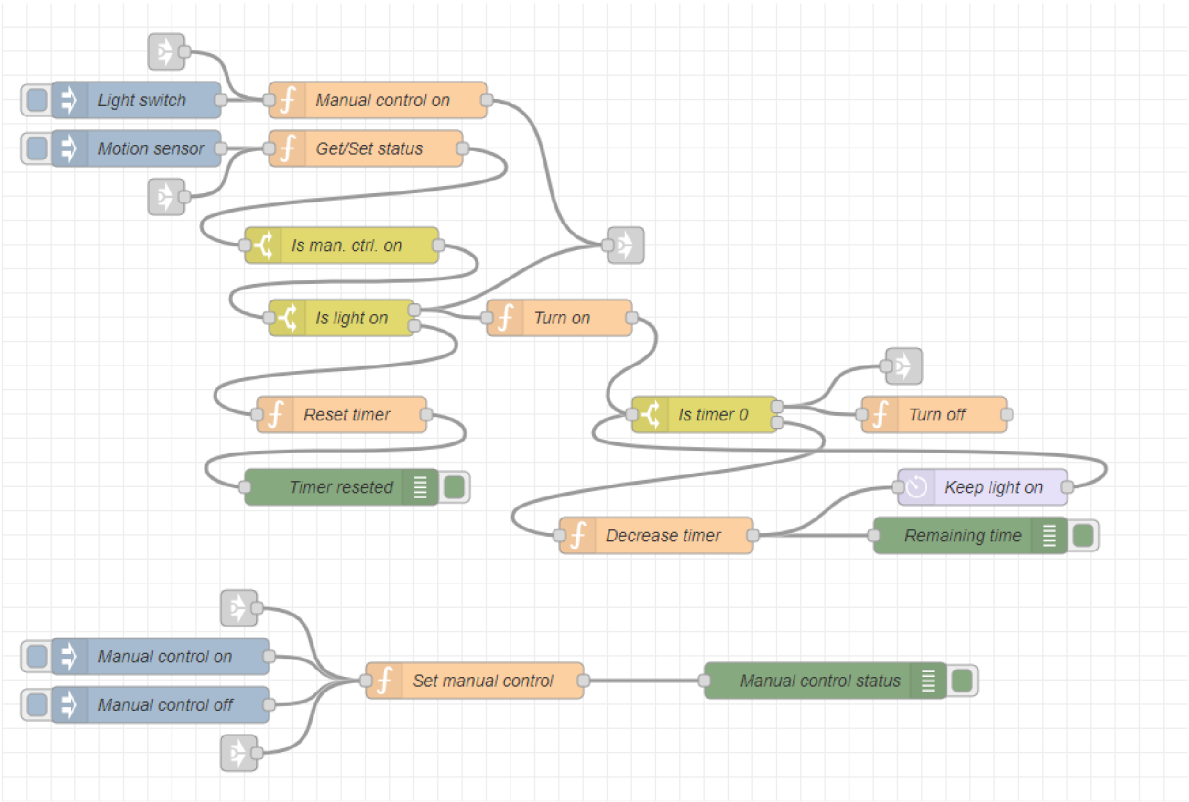

Figure 6.1.3. Motion activated light flow implementation

The *Light switch* input node functions the same as in the basic light flow, only this time, it also sets a flow attribute of manual control to *On* value. That is done in the *Manual control on* function node and serves to prevent conflicts between the light switch and the light sensor. Then the message is sent to the basic light flow via the link-out node. The manual control can also be altered by using the *Manual control on* and the *Manual control off* input nodes. These set the flow attribute of manual control to the desired status. The status is then printed in the *Manual control status* node. Last input is the *Motion sensor* input.

Upon triggering this input node, the status of the flow is set in the *Get/Set status* function node. Then follows a switch node checking whether the manual control is off. In such case the flow continues to another switch node, which checks if the light is already on and the timer should be reset, or if the light is off and should be turned on. If the light should be turned on, the flow forks and one signal is sent to the link-out node connected to the basic light flow and the other signal leads to the *Turn on* function node, where a timer value is set to ten, meaning the countdown has started. After that, the flow reaches the *Is timer 0* switch, which in case the timer attribute value equals zero, and the flow forks once again, one signal leading to the *Turn off* function node,

and the other leading to a link-out node connected to the basic light flow. In the other case, the flow then continues to delay node, where the flow is delayed for one second and then continues to the *Decrease timer* node. There, the timer attribute value is decremented by one, the debug node prints out the remaining time and the flow goes back to the *Is timer 0* switch node.

If the light was already turned on and the motion sensor has been triggered again, the situation is handled in the *Is light on* switch node. In this case the message is sent to the *Reset timer* function node, where the timer is reset, and the debug node informs the user.

6.1.4. Patio light flow

The last instance of a light operating flow is the patio light flow. It is a light which is activated during set hours and deactivated after. Such lights are common in modern homes as outdoor lights, commonly being used on patios and around the house entrance. Like the motion activated light flow, the patio light flow is connected to the basic light flow, so the basic functionality of a light is handled by the basic light flow, and the patio light flow only maintains whether the light should be on or off.

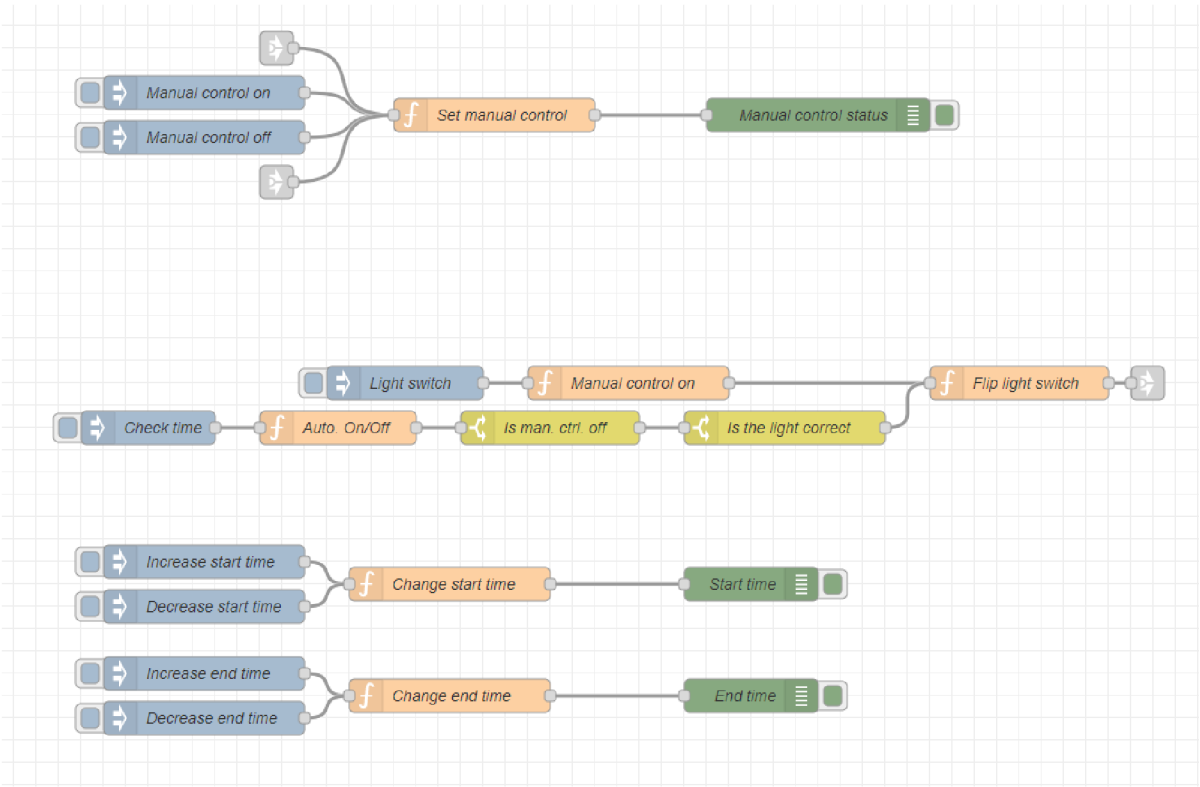

Figure 6.1.4 Patio light flow implementation

The flow is divided to several sub-flows. One of them is designed to handle the manual control of the flow to prevent conflicts. The manual control sub-flow is identical to the one used in the motion activated light flow. The other two sub-flows are to set the time of the light activation and deactivation. The *Increase start time* and *Decrease start time* input nodes lead to the *Change start time* function node, where the start time attribute is set, the default being six o'clock in the evening with the minimum and maximum of twelve o'clock PM and twelve o'clock AM, respectively. Both the start time and end time are printed out in their respective debug nodes. The *Increase end time* and *Decrease end time* input nodes lead to the *Change end time* function node where the end time is set, six o'clock AM being the default value while 3 o'clock AM and 12 o'clock PM being the minimum and maximum.

The last two inputs are the *Light switch* input node and the *Check time* input node. Both contain a number in their payloads used in the basic light flow. If the *Check time* input node is activated, the flow begins, and the message reaches the *Auto. On/Off* function node, where the current time value along with the start and end times values is accessed and the values are compared to determine whether the light should be turned on or off. The flow then continues in the *Is man. ctrl. off* switch node, where the manual control flow attribute is checked. If the manual control is not activated, flow can continue to the *Is the light correct* switch node, which compares the current light status to the status it should be in. If these two values differ, then the flow can continue to the *Flip light switch* function node, where the status value are switch from *On* to *Off* and vice versa and finally the flow is sent to the basic light flow via the link-out node. If the flow is activated by the *Light switch* input node, then the message reaches the *Manual control on* function node, where the manual control is activated and then the message continues to the *Flip light switch* function node.

There are multiple possibilities of inputs for this particular flow, such as a light sensor which would substitute the time setting sub-flows. Node-RED offers nodes supporting network interaction and via a http request type node, the user could easily access weather data from local weather webpage and adjust the sunrise and sunset times automatically.

6.1.5. Summary

Lights are perhaps the most important appliances and usually the first ones to be upgraded with different functions and control possibilities. While simple at the first glance, even the light flows could easily become much more complex and robust by implementing features such as different colors or even presets for an entire room

which would contain data about brightness, color, and a list of lights. It is certainly a possibility which could be expanded perhaps in another thesis focused on smart home flows and their implementation in Node-RED, this time offering more complex solutions comparable to those seen in real smart homes of today.

6.2. Alarm module

6.2.1. Introduction

The alarm module is another one of common and significant module we could expect in a modern smart home. It is an important module which correct implementation is vital for the safety of the building's inhabitants. Despite its importance it is relatively easy to implement and to understand.

6.2.2. Implementation

While all kind of alarms exist, the fire alarm and intruder alarm were chosen and as such this flow has two main inputs: the smoke detector and the break-in sensor. Another input is the switch, which we can use to turn on and turn off the break-in sensor. We cannot turn off the smoke detector for safety reasons. Last input in this flow is the shutdown input which, as the name suggests, shuts down an active alarm. The switch input and the shutdown input could be password protected as to prevent a sabotage, however, the implementation of such functions would defeat the purpose of this thesis to present easy to implement and understand flows.

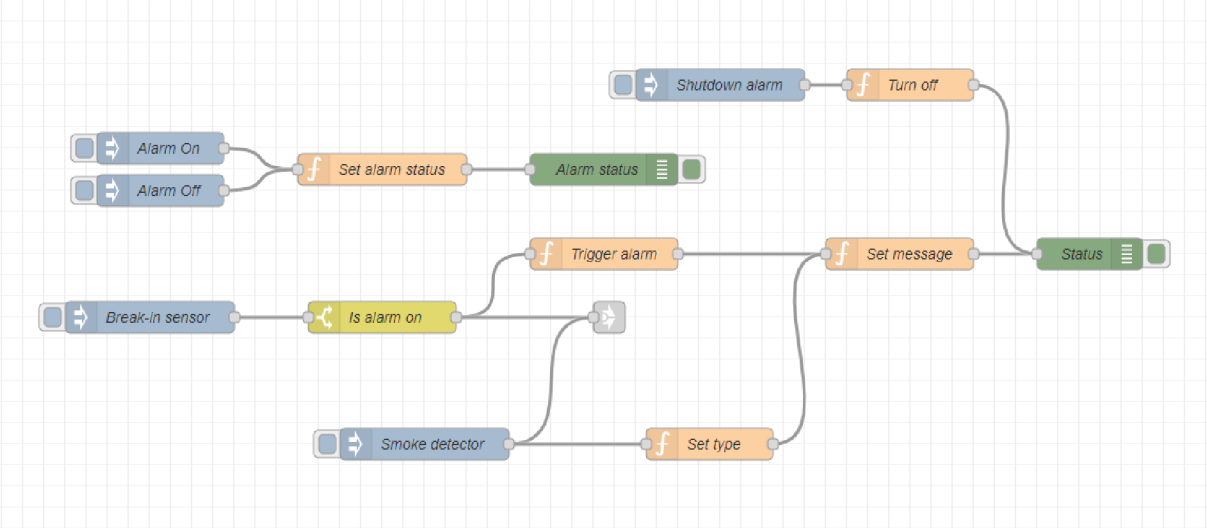

Figure 6.2 Alarm flow implementation

Upon triggering the break-in sensor, the flow begins. First it hits a switch node, where it asks whether the alarm system is already triggered. This serves to block other messages and prevent the alarm from being triggered repeatedly. Then it follows to another switch node *Is alarm on*, where it checks if the alarm attribute is set to *On*. If it is, it forks and one signals continues to the light flow, where it turns on the light at maximum brightness and the other continues in the alarm flow. The connection to the light flow is handled by a link out node. It then hits a trigger alarm node where the flow trigger status is set to *On,* and the alarm type is set to *Intruder*. After that it continues to a *Set message* function node, where the message is set to *<Type> alarm triggered*. Type in this case is set to *Intruder*. Finally, it hits the debug node, which represents the sound device of the alarm. The debug node prints out the message set in previous node and the flow ends.

If the flow is activated via the smoke detector input node, its progress is very similar. The flow skips the initial switch node because the fire alarm cannot be turned off. Instead, it forks right away, and one signal is sent to the light flow while the other continues to a *Set type* function node, where the alarm type and trigger status are set, similar to the *Trigger alarm* function node. At this point the flow's progress is identical as if it were turned on by the break-in sensor. The flow proceeds to the *Set message* node and finally to the *Status* debug node, where it once again prints out the message, which in this case is *Fire alarm triggered*.

To shut down the triggered alarm, the *Shut down alarm* node has to be activated, which sends a message to the *Turn off* node. The *Turn off* node sets the triggered status to *Off* and the alarm is shut down. Finally, to turn on the alarm, or turn it off, the *Alarm On*/*Alarm Off* input nodes should be used. These sent their signals to the *Set alarm status* which set the flow alarm status to *On* respectively *Off*. After that, the debug node called *Alarm status* tells the information whether the alarm is turned on or off.

6.2.3. Summary

This flow, while easy to implement, could easily become very robust by implementing other, quite practical, features, such as other types of alarm. That would of course depend on the location of the building. Other types of alarms could be, for example, a tornado alarm, which could receive tornado warnings via http requests, or a flood alarm, whose functionality would be like the fire alarm's. Cooperation with the blind controlling flow could also be implemented and would be used to clear the escape path for the building's inhabitants. As was mentioned before, such features, while useful, would quickly enlarge the flow and would defeat the purpose of this

thesis, which focuses on presenting easy to implement and easy to understand preset flows. Additional features can always be implemented by a user.

6.3. Temperature control module

6.3.1. Introduction

The temperature control module was quite difficult to implement since it relies heavily on outer influence, which is hard to mimic. If a room has been heated to set temperature, the temperature will not stay stable forever, but will either rise or decline, depending on the weather outside. Due to this fact, this implementation of this module is slightly more abstract, as it had to simulate these external effects. Nevertheless, it is another vital module which should be present in every smart home.

6.3.2. Implementation

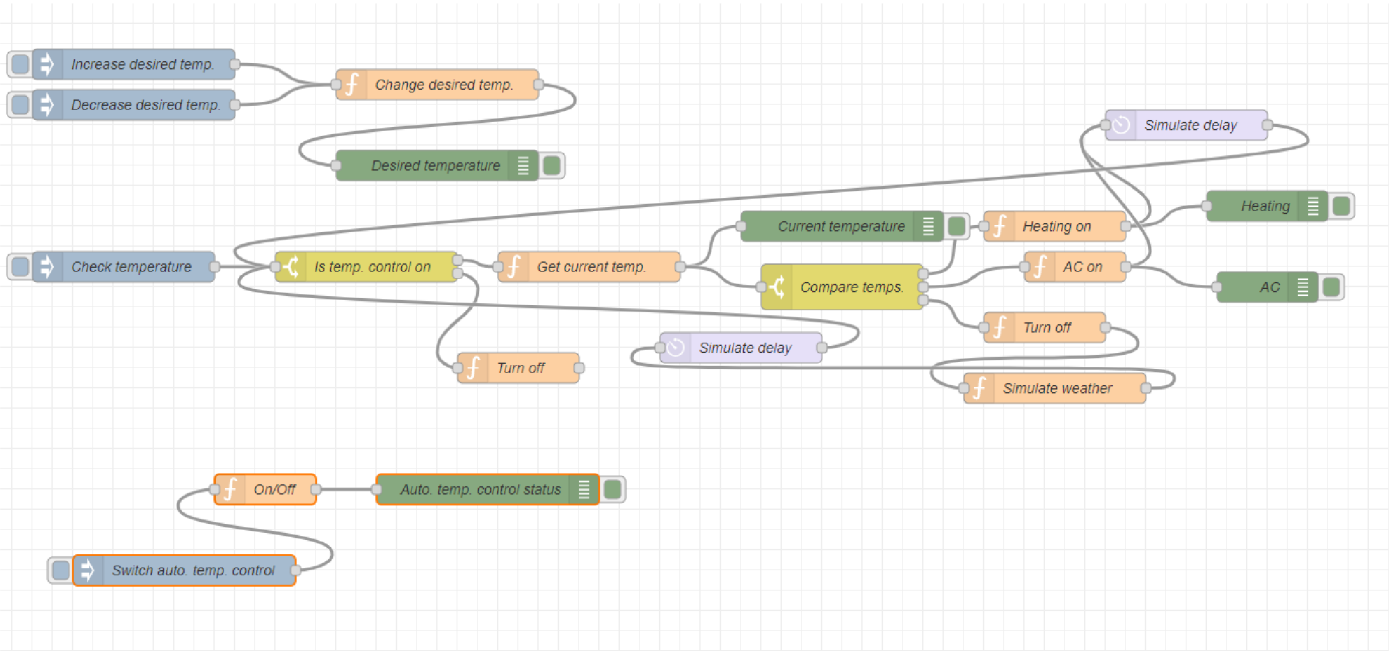

Figure 6.3 Temperature control flow implementation

The temperature control flow has multiple inputs. First, the most important one, is the switch of the automatic temperature control. By activating this input, the automatic temperature control is either turned on or off. The "Auto. temp. control status" debug node then prints out the control status. Other inputs are the *Increase/Decrease*

desired temperature inputs. Upon activating these inputs, the desired temperature attribute will be increased or decreased by one degree Celsius. The *Desired temperature* debug node again serves just to print out the attribute status. The last input is the most important input of this flow and that is the *Check temperature* input node. It simulates a thermometer and sends out a message payload containing a number representing the temperature detected.

When the *Check temperature* node is activated, the flow then proceeds to a switch node, where the automatic temperature control status is checked. If the control is activated, it will then compare now measured temperature to the set desired temperature attribute. Then follows another switch node called *Compare temps.* where the flow forks depending on the values of desired and current temperature. If the current temperature is lower, it activates the heating. The debug node *Heating* representing the actual heating element prints out that the heating is currently on and the temperature is raised by one degree Celsius. The flow then reaches the *Simulate delay* node, which is just an auxiliary node to improve readability of the debug messages. After the second long delay, the flow goes back to the first switch node, where it asks if the temperature control is still on.

If the measured temperature is higher than the desired temperature, the flow is nearly the same, except when it reaches the *Compare temps.* switch node, it will not turn on the heating but instead the air conditioning. Instead of raising the temperature it will lower it by one degree Celsius and the debug node called *AC*, representing the real air conditioning, will print out *AC: On* message. After the one-second-long delay, it goes once again back to the first switch node.

Last situation occurs when the measured temperature is equal to the desired temperature. In such case, upon reaching the *Compare temps.* switch node, it continues to the *Turn off* function node, which turns off both the air conditioning and the heating. Then follows the *Simulate weather* function node, which is more of an abstract node that either lowers the temperature by one degree Celsius, if the heating was on, or raises the temperature, if the air conditioning was on. After that it reaches another *Simulate delay* node and then finally returns to the initial *Is temp. control on* switch node. The flow is cycling and is trying to keep the temperature at the desired level and can be stopped using the temperature control sub flow. When the temperature control is switched off, the flow reaches a *Turn off* node, which turns off heating and air conditioning.

6.3.3. Summary

This preset flow was difficult to implement due to already mentioned fact, that in real implementation it is affected by external influences. Despite that, this implementation could be provided with actual temperature measuring devices and could then be restructured to a working temperature managing flow.

6.4. Shading module

6.4.1. Introduction

This module focuses on the implementation of window shades. The automation of window shades is becoming increasingly more common and as such can be considered as one of the most common flows in smart homes. This implementation features two flows very similar in functionality.

The first flow is more detailed and has delay nodes built in, which were introduced to better simulate actual moving objects. However due to possible conflicts arising from using this flow as the core flow used by various shading objects stored in an array like the light objects used in the basic light flow, a second flow has been implemented which no longer has delay nodes and values change instantly, rather than gradually over time.

6.4.2. Advanced shading flow

The advanced and more detailed flow features five inputs. This number at the first glance might appear too big but four of these inputs represent the raising/lowering and opening/closing of the window shades. The fifth input is a wind sensor whose function is to raise the shades in case of a strong wind to prevent damage. The first four inputs could be easily placed on a small sized touch screen while the fifth input, the wind sensor, would be placed outside and would not be operated by the user.

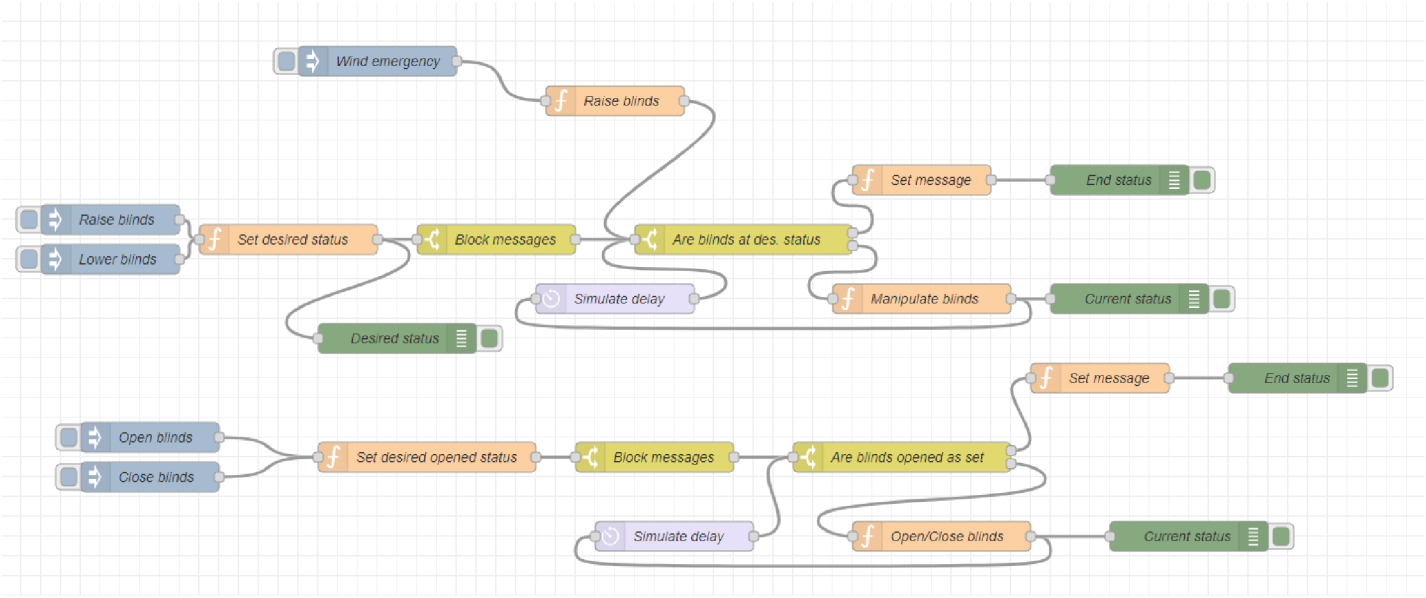

Figure 6.4.2. Advanced shading flow implementation

The advanced shading module are essentially two flows, one designed to raise and lower the shades and the other to open and close them. If the *Raise blinds* or *Lower blinds* input node is activated, then first the desired value attribute is set. Initial value is one hundred, meaning the blinds are fully raised. This value can be either raised or lowered by the increment of twenty-five with zero and one hundred being the minimal and maximal values. Desired value is then print out by the *Desired status* debug node and the flow continues to the *Block messages* switch node, which serves to block additional messages if the flow is already active.

Following this switch node is another switch node called *Are blinds at des. status* which simply checks whether the current status of the flow is equal to the expected status. If the values are different, the flow heads to the *Manipulate blinds* function node, whose function is to either raise or lower current status attribute value by the increment of five. The *Current status* debug node prints out the current status and the *Simulate delay* node halts the flow for half a second to simulate real functionality. When the current status equals the desired status, the flow then proceeds to the *Set message* function node. There, the message payload attribute is set to contain information about the current level of blinds and this message is printed out in the *End status* debug node.

This sub-flow also has the *Wind emergency input* which sets the desired attribute value to one hundred and then proceeds to the *Are blinds at des. status* switch node,

effectively fully raising the blinds and preventing them from being damaged by dangerous wind.

The sub-flow opening and closing the blinds is very similar to the former sub-flow. It has two inputs, *Open blinds* and *Close blinds*. Both inputs lead to the *Set desired opened status* which also increases or decreases the desired attribute value by twenty-five and then the flow continues to the *Block messages* switch node, identical to the switch node in the previous flow. If the flow is not already running, the message reaches the *Are blinds opened as set* switch node where, if the desired and current attributes aren't equal, the flow goes to the *Open/Close blinds* function node, where the current value is either raise or lowered by the increment of five and then in the *Simulate delay* node the flow stops for half a second and then reaches the *Are blinds opened as set* switch node again. If the current and desired values are equal, the flow reaches the *Set message* node where the message payload is once again set, and its content print out in *End status* debug node.

6.4.3. Simplified blinds flow

The simplified flow has no delay nodes and therefore has no need for two attributes to hold values of current and desired status. Instead, it only has current status. However, the key difference between the advanced and simplified flows is the numeric value contained in the message object sent by inputs. Like the light module, before using the simplified flow the array holding the objects must be initialized using the *Init./Reset blinds* input node.

When the array is initialized, the simplified flows can be used. The *Raise blinds* and *Lower blinds* inputs lead to the *Manipulate blinds simplified* function node, where the numeric value contained in the message object is used to access the array of objects. The objects current status is then either raised or lowered by the value of twenty-five and the message payload is set to contain the information about the object. As usual, the message is then printed out in the debug node.

The *Open blinds* and *Close blinds* inputs are nearly identical; except they lead to the *Open/Close blinds simplified* where the opened status attribute is changed based on the input. The message is printed out in the *Blinds status* debug node. The wind emergency in this flow functions similarly to the alarm in light module as it sets the raised status of all objects in array to one hundred.

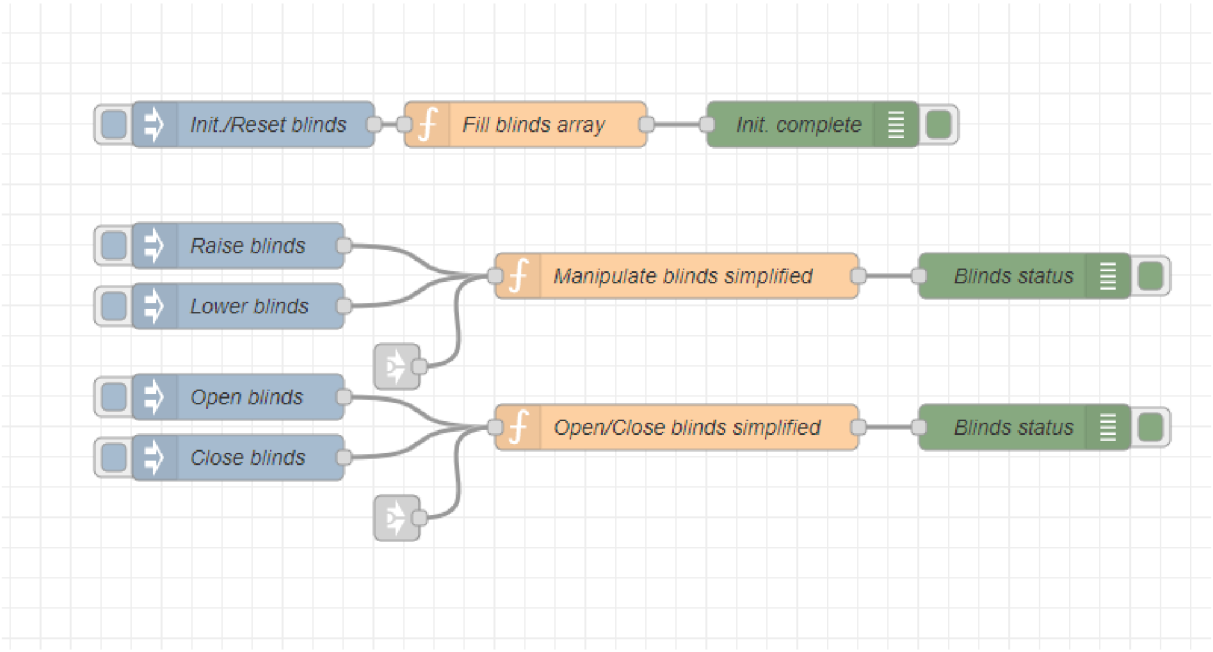

Figure 6.4.3. Simplified shading flow implementation

6.4.4. Summary

This module, like the heating module suffered from the usage of basic nodes and the unavailability of more powerful and advanced nodes. The problem is in the delay nodes as they cannot handle multiple messages passing through and in the switch nodes, which have only limited capability of values they can control. However, despite the need for a simplified version of this module, its implementation can still be considered successful.

6.5. Smart home implementation

The smart home implementation flow is a series of inputs connected to various previously described flows. Its purpose is to prove the functionality of flows and their independence, meaning various instances of a device can be connected to and use the same flow without any conflicts or issues.

The smart home flow features eight different lights, with one being connected to the motion activated light flow, four window blinds, one alarm control along with three break-in sensors and one smoke detector and one set of temperature control inputs. The flow also features two auxiliary inputs to initialize the arrays in shading and light modules. When a certain input is activated, the debug output window shows all information about the flow. In case of *Light 3* input, for example, the debug message will contain the light status, brightness and room (assigned in initialization).

To set up a custom background for the flow proved to be too difficult to implement and as such, inputs are not put on the background of an actual house plan. This means Node-RED is not an ideal tool for the house-planning, but it is still a tremendously useful tool for designing and customizing smart home flows.

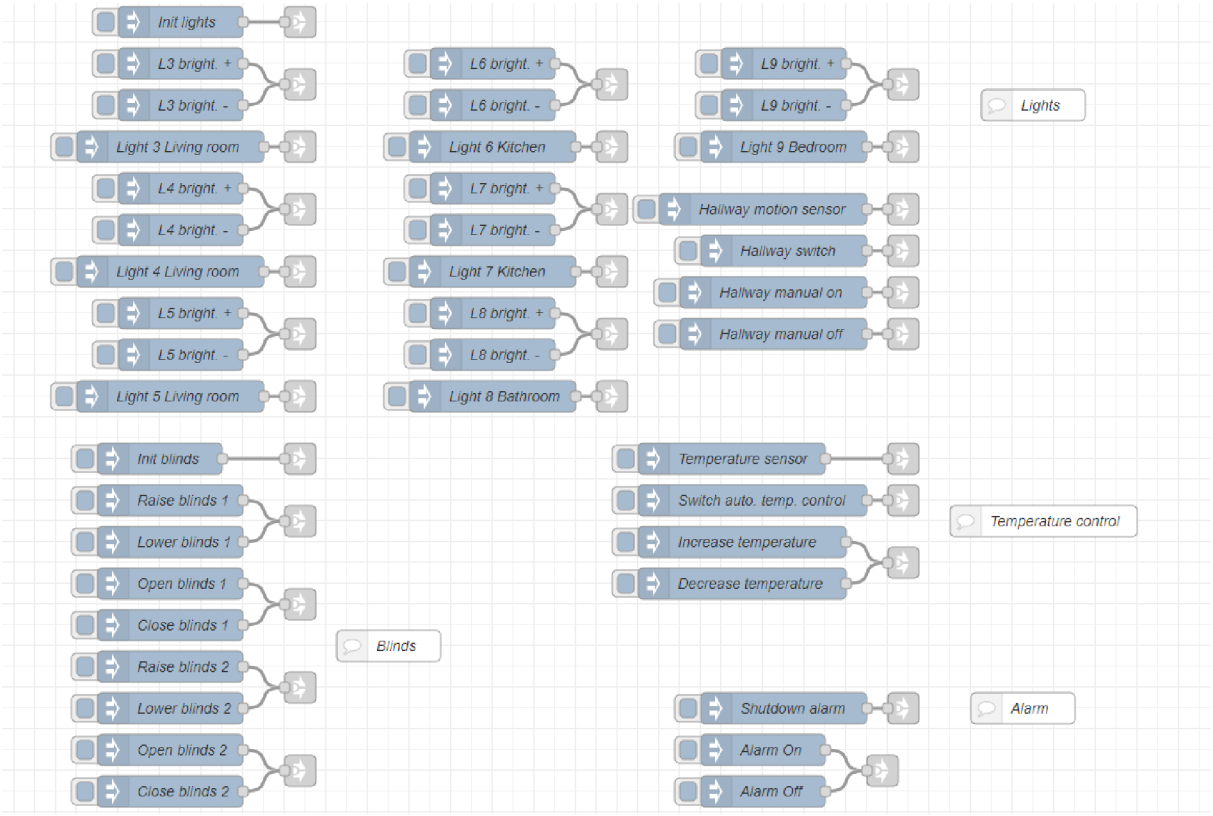

Figure 6.5 Sample of smart home inputs

7. Final conclusion

The goals of this thesis were to analyze the existing smart home solutions, describe the most common modules and to implement them along with a model smart home in Node-RED. The commercial solutions were listed and described, and the most common modules were identified.

The contribution of this thesis was the introduction of the Node-RED development tool, its usage in the smart home environment, and the implementation of the preset flows which can now be used as a foundation or an example for designing smart home flows. These preset flows can now be downloaded or previewed and customized to fit their roles as precisely as possible. This is vital in the smart home development field and provides a new and free way of implementing smart homes without relying on expensive commercial solutions.

So far, the only possibilities were either to order costly bespoke smart home modules, which offered a large number of options, but were required to be implemented by certified company specialist. The other option is buying expensive brand devices and connecting them to the central unit, which offered a simple and effortless installation ways and the overall comfort at the cost of customizability. By presenting the option of using Node-RED and preset flows, these two options combine into one offering advantages of both options and yet holding the negatives.

Using the Node-RED development tool, simple foundation flows were implemented, and their functionality described along with possible future room for upgrades. The topic of this thesis can surely be expanded by using more advanced Node-RED features, such as the dashboard, or more powerful nodes utilizing API requests from weather pages and implementing a database. Another possible extension of this topic could be connecting implemented flows to a phone device and finally to a Raspberry Pi and use Node-RED to operate a device.

Even without these functions implemented, the simple preset flows offer tremendous possibilities to people interested in smart homes with basic technology knowledge and even to people without such knowledge, who want to learn.

8. Bibliography

[1] N. Heath, "www.techrepublic.com," 13 3 2014. [Online]. Available: https://www.techrepublic.com/article/node-red/. [Accessed 10 5 2021].

[2] J. CHEN, „www.investopedia.com,“ 25 2 2020. [Online]. Available: https://www.investopedia.com/terms/s/smart-home.asp. [Přístup získán 15 5 2021].

[3] „www.statista.com,“ 6 2020. [Online]. Available: https://www.statista.com/statistics/1201156/united-states-smart-home-device-awareness/. [Přístup získán 20 5 2021].

[4] T. Perry, „www.smarthomepoint.com,“ 27 3 2020. [Online]. Available: https://www.smarthomepoint.com/alexa-echo-difference/. [Přístup získán 15 5 2021].

[5] "https://apps.apple.com/," Apple, [Online]. Available: https://apps.apple.com/us/app/amazon-alexa/id944011620. [Accessed 15 5 2021].

[6] PYMNTS, „www.pymnts.com,“ PYMNTS, 13 5 2019. [Online]. Available: https://www.pymnts.com/amazon-alexa/2019/amazon-alexa-digital-assistant-popular/. [Přístup získán 17 5 2021].

[7] D. Norris, Home Automation with Raspberry Pi: Projects Using Google Home, Amazon Echo, and Other Intelligent Personal Assistants, McGraw-Hill Education TAB, 2019.

[8] LIDL, „www.lidl.cz,“ LIDL, [Online]. Available: https://www.lidl.cz/smart-home. [Přístup získán 17 5 2021].

[9] „/www.loxone.com,“ Loxone, [Online]. Available: https://www.loxone.com/cscz/chytry-dum/. [Přístup získán 12 5 2021].

[10] „nodered.org,“ 14 6 2016. [Online]. Available: https://nodered.org/blog/2016/06/14/version-0-14-released. [Přístup získán 14 5 2021].