TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

German Novikov 112587IAPB

UNIFICATION OF SMART HOME COMPONENTS NETWORK COMMUNICATION USING NODE-RED

Bachelor's thesis

Supervisor: Enn Õunapuu
Professor

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

German Novikov 112587IAPB

ARUKATE KODUKOMPONENTIDE VÕRGU KOMMUNIKATSIOONI UNIFITSEERIMINE, KASUTADES NODE-RED-I.

Bakalaureusetöö

Juhendaja: Enn Õunapuu
Professor

Tallinn 2018

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: German Novikov

20.05.2018

Abstract

At present, there are many components of a Smart Home provided by different brands. Each company that produces these components usually uses its own standards. Thereby, because of the different standards of the components used, it is often not possible to unite all Smart Home devices into one large network.

This thesis describes a proposed solution to this problem in the form of using a Node-RED application that allows to connect various components of a Smart Home (which rely on different standards) with each other with minimal application configurations required.

To test this application, a simulation of the Smart Home was written using Java programming language.

This thesis is written in English and is 38 pages long, including 4 chapters, 9 figures and 1 tables.

Annotatsioon

Arukate kodukomponentide võrgu kommunikatsiooni unifitseerimine, kasutades Node-RED-i.

Tänapäeval pakuvad paljud firmad erinevaid targa kodu komponente. Iga firma, kes tegeleb nende komponentide tootmisega, kasutab tavaliselt oma standardeid. Erinevate kasutatavate komponentide standardite tõttu ei ole tihti võimalik ühendada targa kodu seadmeid ühte võrku.

Antud lõputöö kirjeldab pakutud lahendust sellele probleemile: kasutada Node-RED rakendust, mis võimaldab ühendada erinevaid targa kodu seadmeid (mis kasutatavad erinevaid standarte) omavahel nii, et nõutud rakenduse seadistamine on minimaalne.

Pakutud rakenduse testimiseks loodud targa kodu simulatsioon on kirjutatud Java programmeerimise keeles.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 38 leheküljel, 4 peatükki, 9 joonist, 1 tabelit.

List of abbreviations and terms

Node-RED	is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways. It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click. [1]
HTML	Hypertext Markup Language - is the standard markup language for creating web pages and web applications. [2]
MQTT	Message Queue Telemetry Transport - is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. [3]
XML	Extensible Markup Language - is a markup language for documents containing structured information. [4]
JSON	JavaScript Object Notation - is a lightweight data-interchange format. [5]
URI	Uniform Resource Identifier - is a string of characters designed for unambiguous identification of resources and extensibility via the URI scheme. [6]
URL	Uniform Resource Locator is a reference (an address) to a resource on the Internet. [7]
HTTP	Hypertext Transfer Protocol is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. [8]
TCP	Transmission Control Protocol is a connection-oriented reliable protocol. It provides a reliable transport service between pairs of processes executing on End Systems (ES) using the network layer service provided by the IP protocol. [9]
UDP	User Datagram Protocol - is a long standing protocol used together with IP for sending data when transmission speed and efficiency matter more than security and reliability. [10]
Java	Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically

designed to have as few implementation dependencies as possible. [11]

Table of contents

List of figures

List of tables

1 Introduction

1.1 Background and statement

Today, there are many different devices that can be used in the Smart Home system. Each company that offers components for the Smart Home has its own communication standards and often there is no way to combine such components into one system, due to the fact that they use different communication protocols, which are incompatible with each other by default. Thus, in order to build a Smart Home, people have to buy devices from one company, search for companies with similar standards, or write the complex programs which would allow to combine such components

In the Smart Home market there are many ready-made complex solutions for building a Smart Home. Such solutions are suitable if client has specific needs and in the future is not going to add devices from other solutions that cannot interact with already installed system. If client happens to want to add a new device which is built using other standards, he will have to write (or order) a program that will allow him to add a new device to the existing system in accordance with the differences between communication protocols, i.e. the program will play a broker role and handle the communication between components, transforming requests and responses into appropriate format.

1.2 Goals and methods

This thesis is intended to solve the problem of building Smart Home based on many different components, which communicate with each other by means of different protocols.

To solve this problem, the Node-RED programming tool will be used. Node-RED allows to create a connection to various devices in one system and to establish rules for their communication.

In this work, a simulation of the Smart Home will be built. In which Node-RED will perform the tasks of a broker between devices and servers. It will also assume the responsibility of managing the entire system.

That will demonstrate that the use of the Node-RED programming tool allows to solve this kind of problem in the most convenient way, without writing complex programs and with no restrictions related to the choice of Smart Home components provider.

2 Main Results

2.1 General description of the proposed solution

The one of main goals of the proposed solution is to simulate a Smart Home system. In the provided simulation the role of sensors and control elements performs the applications written in Java, which is running on the Tomcat [12] server. Sensors and controllers communicate with each other through various protocols. The application written using Node-RED programming tool controls and connects all the components of the Smart Home application. This broker application tracks the changes that occur on the sensors (simulated) and performs the actions in accordance with the received data.

Also some actual online web services are connected to the Node-RED application, which allows the simulated system to obtain real data from the internet from third-party service providers.

2.2 Smart Home simulation

2.2.1 Setting up environment

- Install version 8.10.0 of Node.js [13], it will be used as base for Node-RED.
- Install version 0.18.4 on Node-RED to the working directory.
- Install Java 8 for simulating sensors and home controllers.
- Install Tomcat server version 8.0.321 for running simulation of sensors and controllers.
- Install Mosquitto [14] version 1.4.15a as server for MQTT protocol
- Google Chrome version 66.0.3359.139 was used as an environment to run the Node-RED application.

2.2.2 Simulation of sensors

To simulate behaviour of the Smart Home sensors a separate application was built. There are four Java classes in the application, which send data of different format using different

communication protocols to the installed Node-RED server URI endpoint "http://127.0.0.1:1880". Class and sensor identifier are being added to each URI request sent from the application.

Each class performs the role of the room in which the sensors are located. This application is written in the Java programming language and installed on the Tomcat server.

Each class (room) includes four sensors:

- **Temperature sensor** which sends data in Celsius degree. Temperature is being randomly generated within range from 15 to 35 °C.
- **Humidity sensor** which sends data in percent's. Humidity value is being randomly generated within a range from 30 to 70 %.
- **Illumination sensor** which sends data in Lux. Illumination is being randomly generated within 0 to 500 Lux.
- **Switch sensor** which sends data as 1) "0" or "1" where "0" means "switched off" and "1" - "switched on". 2) If given class represents cabinet, then value is being sent as Boolean and can be true or false respectively.

In the table below are provided names of the protocols which are used by the sensors of each room along with the data format in which messages are being sent from the sensors to the Node-RED server URI endpoint.

Table 1 Communication protocols and message data format used by the sensors in each room

Room	Used protocol	Data format
Bedroom	HTTP Post Request	JSON Format
Cabinet	HTTP Post Request	application/x-www-form-urlencoded
Kitchen	HTTP Post Request	XML Format
Garage	MQTT	JSON Format

Sensors identifiers are being attached to the request address and are as follows:

- temperature
- humidity
- illumination
- switch

Detailed description of the communication between Java classes and the Node-RED server (each header below represents the exact name of the corresponding Java class):

- **BedroomSensorsContainer:**

  This class sends data to URI endpoint “http://127.0.0.1:1880/bedroomHandler/” plus identifier of sensor. Data is being sent as POST HTTP Request. The content is being added to the request body in JSON format. (Code for this class is provided in the appendix 1.)

  Data format example:

  ```
  {“parameterName”:”HUMIDITY”, “value”:”35”}
  ```

- **CabinetSensorsContainer:**

  This Class sends data to URI endpoint “http://127.0.0.1:1880/cabinetHandler/” plus identifier of sensor. Data is being sent as POST HTTP Request. The content is being added to the request body in “application/x-www-form-urlencoded” format.

  Data format example:

  ```
  HUMIDITY=35
  ```

- **KitchenSensorsContainer:**

  This class sends data to URI “http://127.0.0.1:1880/kitchenHandler/” plus identifier of sensor. Data sends as Post HTTP Request where adding in to body data in XML format.

  Data format example:

```
<parametrName>HUMIDITY</parametrName><value>35</value>
```

- **GarageSensorsCpntainer:**

  This class sends data by MQTT protocol to mosquitto server, from which the Node-RED application requests the data. Mosquitto server can be reached by address “tcp://localhost:1883”, and the topic for message that is being sent is a sensor name. Data is being sent in JSON Format.

  Data format example:

```
{“parameterName”:”HUMIDITY”, “value”:”35”}
```

2.2.3 Node-RED receiver

The Node-RED allows to receive data using various communication protocols. Connection settings are provided by nodes which are specially created for handling specific connections. That feature allows to set up a Smart Home system in which the modules communicate with each other using different communication standards, as there is possibility to explicitly set up individual node for each of the corresponding standards.

In this work such nodes are as follows:

- “http”;
- “mqtt”
- “websocket”
- “tcp”
- “udp”

The diagram (Figure 1) below is based on the graphical user interface of the Nod-RED programming tool. It represents the process of obtaining and validating the data, and also assigning values to the global variables based on the received data.

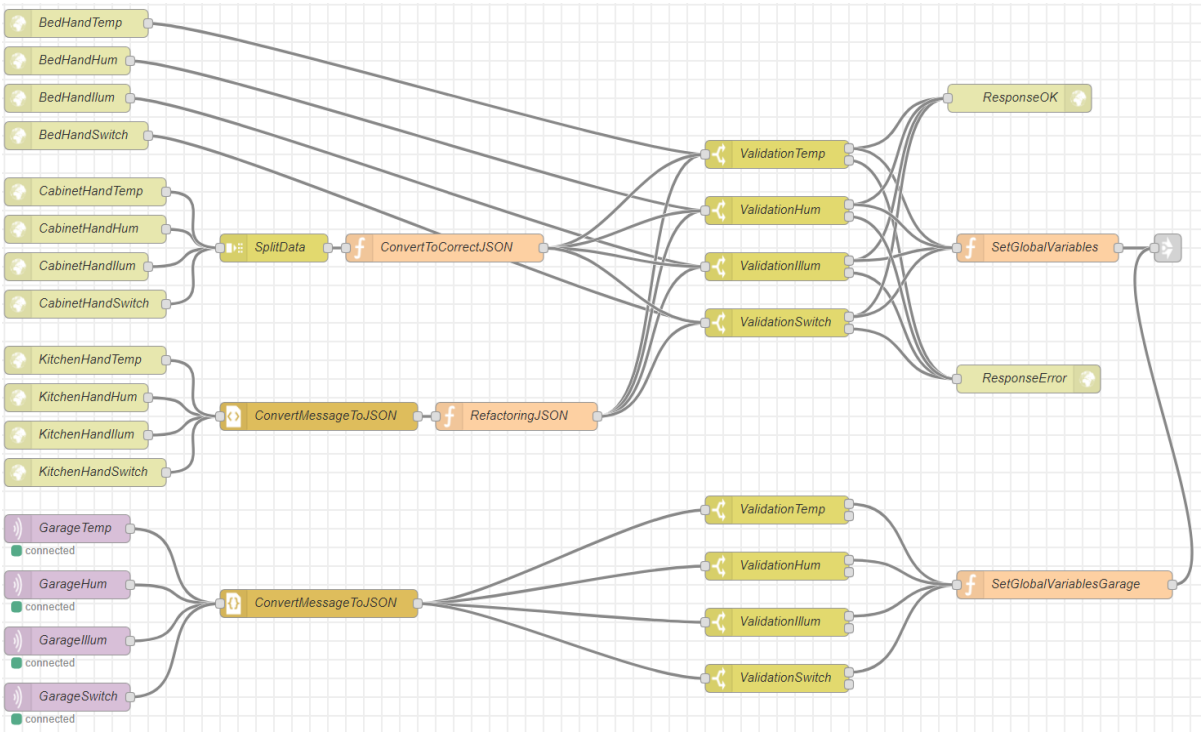


Figure 1 Node-RED receiver scheme

Node-RED has node for receiving HTTP requests. This node performs all the work on receiving and processing requests. To obtain the request in the Node-RED, only two fields of the node should be set. The first field is a “Method” field, which contains the data about the HTTP request method (POST/PUT/DELETE/GET/PATCH). The second field is an “URL”, it contains the HTTP path which should be used to receive requests. These two fields are enough to receive a HTTP Request.

To make scheme more readable for the user, there is one more optional field which helps to identify this node - “Name” field. The name of the node (if any) should be provided here. Field “Name” is applicable for every node in Node-RED, as it helps to find the needed node easily. The “Name” is also being shown in logs produced by Node-RED application.

For the correct work of the HTTP Request node the flow should send a valid response. For that purpose, separate node called “http response” should be added to the flow. The field “Status code” of response node contains code of the response, indicating the response state and applied based on the request validation results. There are two codes used in this work:

- First code is “200”, indicating that request was correct.

- Second response code is “500”, indicating that something was wrong with the request.

Codes were selected accordingly to the HTTP standard [15]. Other response codes (e.g., custom ones) can be used as well, but are out of the scope of the given work.

To receive data using the MQTT standard, a special “mqtt” node is used which allows to quickly set up a connection. Four configured nodes connect to the localhost:1883 server and wait for messages from sensors. Each node is configured for a specific sensor. Names of the sensor nodes are provided below:

- GarageTemp
- GarageHum
- GarageIllum
- GarageSwitch

After the message is received, its data should be converted correspondingly to be presented in the one standard across further flows. This is needed for convenient usage of the data in the further operations. To convert the data to the needed standard, the following nodes are used:

- “split” - allowed to separate the data received from the cabinet sensors
- “xml” – allowed to convert data from XML standard to JSON
- “json” – allowed to convert data from JSON string to JSON object
- “function” – allows to create necessary structure from the received data depending on custom requirements

For validation purposes such nodes as "switch" are used, which allow to specify the conditions under which data is sent to the specific node outputs. At this stage of data acquisition, the conditions under which the sensors would send a positive response about the reception of data were validated.

In Node-RED, there are three levels of variables: local, flow, and global. These levels make it more convenient to transfer data between levels. Local variables can only be used in one particular node. If the value of a local variable should be passed to another node, the variable itself should be passed along with the message. The variable flow can be used in all nodes of the flow. Global can be invoked and changed throughout the entire grid.

After data is validated, it is being assigned to the global variables if validation was successful. If validation failed, no assignment occurs and corresponding error is being thrown instead.

To transfer variables, "function" node was used, which allows to write various functions in the JavaScript language, which have access to the Node-RED global variables and also can receive variables as function arguments.

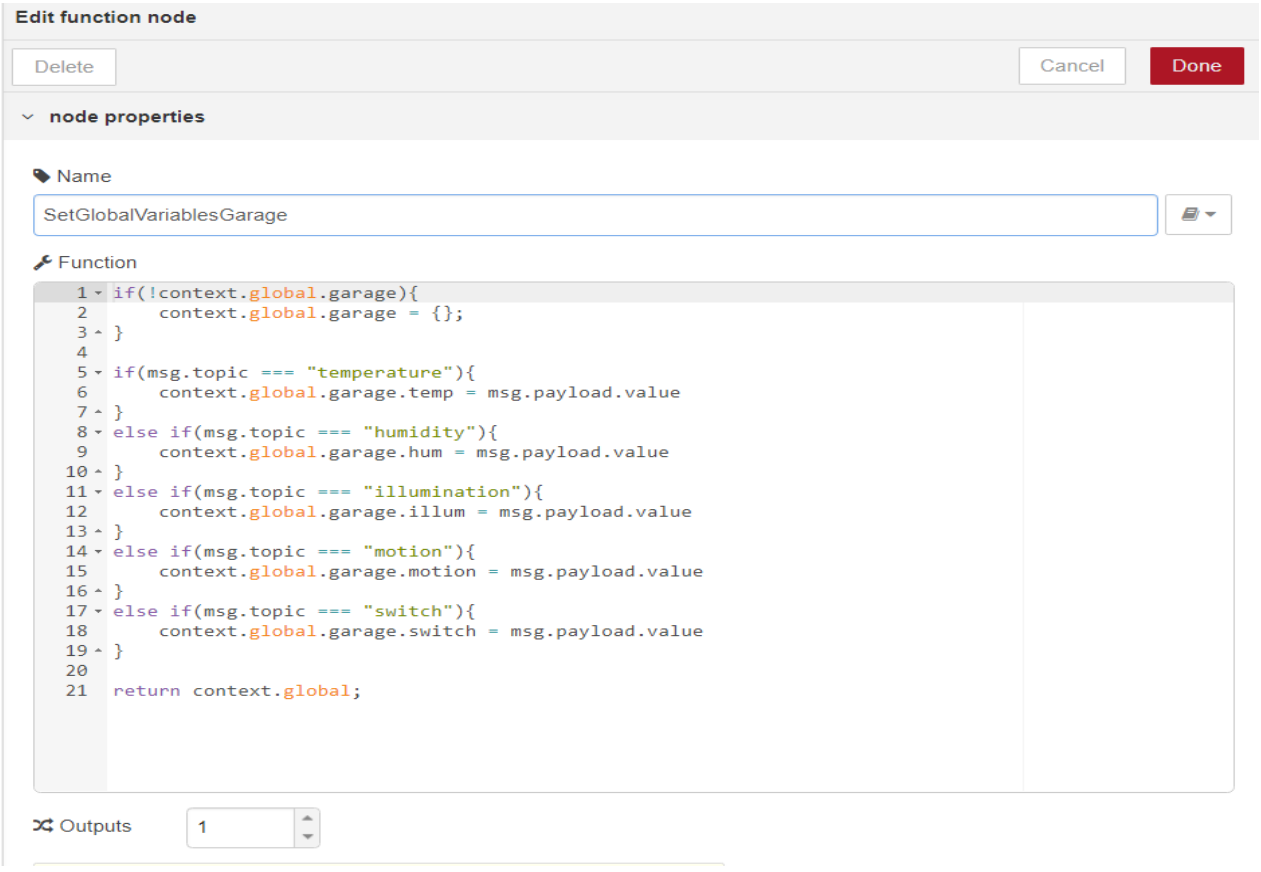


Figure 2 Function adding variables to global

As a result of constructing this flow, data from the sensors was successfully acquired and then checked for compliance, answers were sent to the corresponding sensors and data values assigned to the global variables. After performing these operations, the progress of the task is being redirected to the next flow.

2.2.4 Management system

The next stage of building the Smart Home was to build a lighting control scheme, temperature control scheme and opening / closing ventilation scheme. In order to do that, a new flow was created, which is responsible for managing all systems.

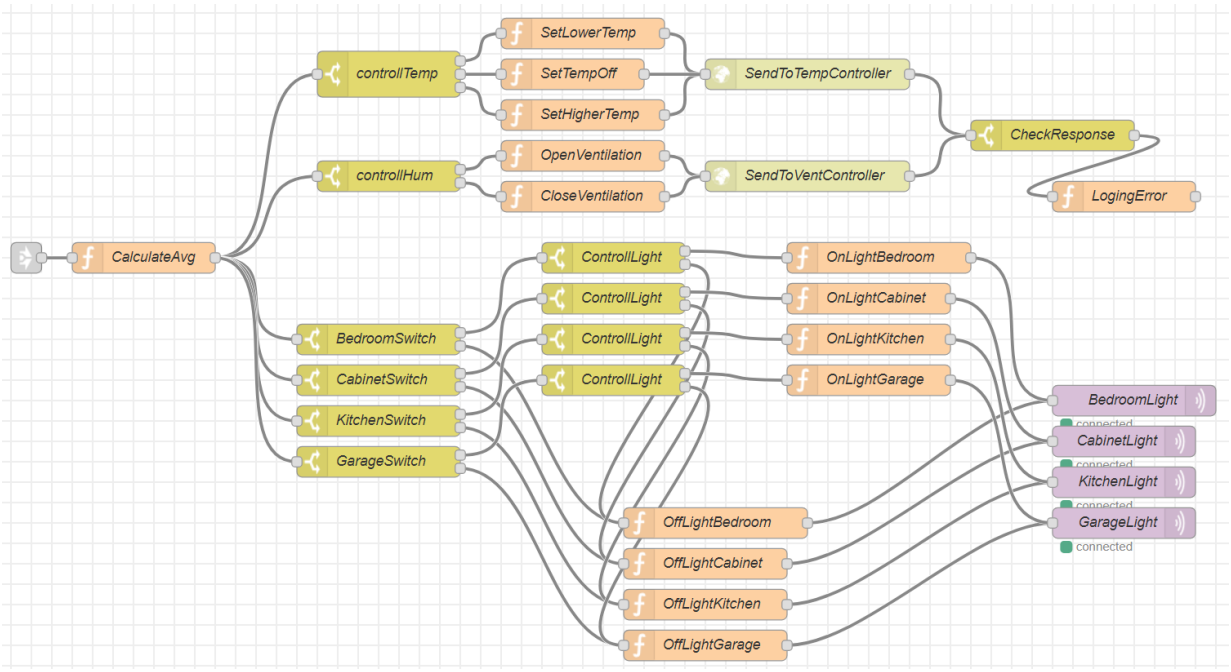


Figure 3 Device management flow

Program switches to this flow after receiving data from the sensors and processing them, which is described in paragraph 2.2.3.

The first thing that should be configured is the name of the parent flow, i.e. the flow, after which the given flow should perform its operations. This is necessary to ensure that these operations are performed after receiving the data from the sensor.

For the following operations, average values of the data received from the house sensors is needed. For that purpose, the function that considers the average values of the temperature, illumination and humidity was written. This function is represented by the "function" node.

After obtaining the mean values, various action scenarios were written. Each scenario for corresponding sensor depends on the obtained results. Scenario can be considered as the actions performed after validation of received data, based on the validation results.

Conditions on which scenarios depend are represented by "switch" nodes, which made it possible to quickly describe the corresponding necessary rules.

To regulate the temperature, three scenarios were implemented:

1. If the average temperature is more than 24 degrees Celsius then Node-RED sends the temperature control system command "lower", which means that temperature should be reduced.

2. the average temperature is between 20 to 24 degrees Celsius then Node-RED sends the temperature control system command "off", which means that the temperature level is appropriate and no actions should be taken right now.

3. If the average temperature is lower than 20 degrees Celsius then Node-RED sends the temperature control system command "higher", which means that temperature should be increased.

To regulate the humidity, two scenarios were implemented:

1. If the average humidity is between 40 to 60 % then Node-RED sends the ventilation system command "off", which means that the humidity level is appropriate and no actions should be taken right now.

2. If the average humidity is NOT between 40 to 60 % then Node-RED sends the ventilation system command "on", which means that the fresh air from the outside should be blown into the room.

To switch the light on, we first check the light switch, if it is in the "ON" position, then we check the illumination in the room. For each room there is a unique level of illumination (threshold) at which the light should be turned on. To control the light, two necessary nodes of the "switch" type were written for each room: first node is responsible for checking the state of the light switch ("ON"/"OFF"), second node is responsible for obtaining the illumination level. The example of light control flow for the bedroom is provided below:

- The first node checks the state of the light switch:

- If the light switch is in the off position, then we immediately send a command to the room lighting system - "off". No further actions are being performed.
- If the light switch is turned on, then the verification process goes to the second node (described below).

- The second node checks the illumination level:
  - If in the bedroom the illumination is below 100 Lux, then "on" command is being sent the lighting system, which means that lights should be turned on automatically, as the illumination level for the given room is considered to be insufficient.
  - If the value exceeds 100 Lux, then "off" command is being sent, which means that lights should be turned off, as the natural illumination level is sufficient for the current room.

The last action in this flow is sending commands to the control systems. In order to send commands, two different protocols are being used: HTTP and MQTT. According to the HTTP protocol, commands to the systems of regulation of temperature and ventilation are sent. Commands for lighting control systems are sent via the MQTT protocol.

HTTP protocol commands are sent to the "localhost:8080/controller/", which are then validated and after that corresponding response code is sent (depending on the validation results). The answer received from the control systems is checked and if the code is not "200" then Nod-RED logs the error code.

Logging occurs in the node "function".

```
node.warn("Response from controller not 200" + msg.statusCode);
```

Figure 4 Logging function

MQTT protocol commands are sent to the "localhost:1883" and then validated. For this protocol there is no answer provided, because of that no further actions needed.

In this flow, the response rules were written, which depend on the data received from the sensors and the action commands were sent to the control systems of the Smart Home in

various ways. In order to implement described logic, the basic nodes of Node-RED were used, which allowed to write the entire flow of processing data and sending different commands using simple components.

2.2.5 Building a dashboard and starting the simulation.

The simulation developed in the scope of the thesis starts working when dashboard page is opened. Once the dashboard window opens, entire simulation runs.

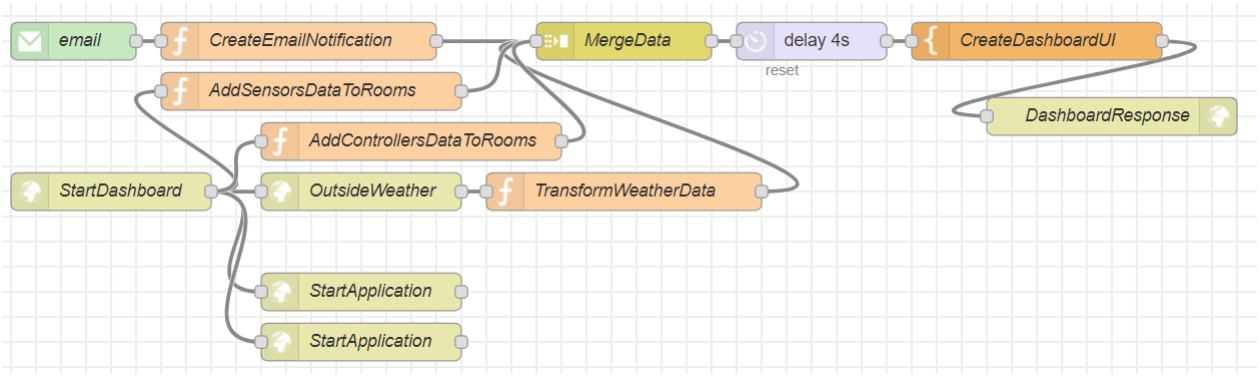


Figure 5 Start simulation flow

The flow depicted on the Figure 5 begins by accepting a request to open a dashboard page. Dashboard page can be accessed by the URL "http://127.0.0.1:1880/dashboard" (in order to this endpoint to be accessible, Node-RED server should be started). After the request received, three actions are performed:

- The request is sent to the URL "localhost:8080/start" to start the sensor simulation, which will send requests with sensor data to Node-RED server.
- The request is sent to the URL "localhost:8080/controller/start" to start the control system simulation, which will receive requests with commands from Node-RED
- The request is sent in order to obtain weather data from the URL "http://api.openweathermap.org/data/2.5/weather?id=862995&appid=2db0efa488fab01338ef94da1d2c1498", which is one of the publicly available weather

service API. The data is obtained as JSON object for Tallinn city.


```
{"coord":{"lon":24.74,"lat":59.47},
    "weather":[{"id":801,"main":"Clouds","description":"few clouds","icon":"02d"}],
    "base":"stations",
    "main":{"temp":294.15,"pressure":1012,"humidity":46,"temp_min":294.15,"temp_max":294.15},
    "visibility":10000,
    "wind":{"speed":4.1,"deg":20},
    "clouds":{"all":20},
    "dt":1526478600,
    "sys":{"type":1,"id":5014,"message":0.0018,"country":"EE","sunrise":1526435038,"sunset":1526496747},
    "id":862995,
    "name":"Tallinna linn",
    "cod":200
}
```


Figure 6 Example of obtained weather data in JSON format

After receiving the data, temperature is converted from Kelvin degrees to Celsius degrees. “function” node is used in order to perform this conversion.

- To perform the conversion, the weather data is transferred from global variables to local variables. The data transfer occurs inside the same "function" node. For data transfer, two different nodes were used to ensure more convenient merging in the next operation.

Node-RED also provides the node which allows to connect to e-mail services. This node is named correspondingly - “e mail” and allows to set a lot of parameters for configuring connection to e-mail service. With the help of this node it is possible to read letters from specific e-mail folder, and, if needed, mark letters as “Read” or delete specific letter. All messages come in JSON format and therefore can be easily used in the following steps.

In this flow the node "e mail" is used to establish a connection with an existing Gmail account. This node is configured in this way, that when new letter is received, it does nothing with the letter state, which allows to collect unread letters.

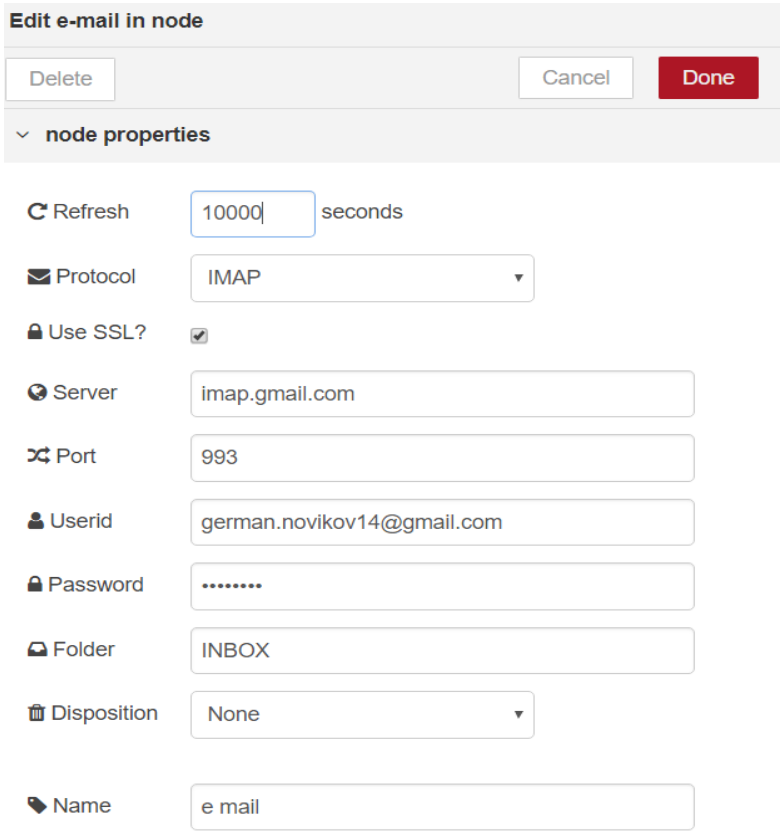


Figure 7 Configuration of the e mail node

After receiving the letter, the necessary information is being gathered. In the purpose of simulation only the title of the letter is needed. For further operations all titles of the unread messages are being added into an array.

For the following steps all data should be collected together. To achieve that, "join" node is being used. This node allows to create a new object from several others. In the parameter you specify the object to take from the received message content and the name that should be assigned to this object. Also number of how many objects should be collected in order to transfer them further can be specified as well as an option to configure the node to wait for the message, which can be achieved using “complete” command.

Due to the fact that data transmission and processing takes some time, it was necessary to set the delay timer for four seconds. Because of this, the data is being simultaneously outputted to the dashboard, which prevents problems with the loading of the elements. In order to delay the transfer of data to the dashboard, the " delay" node is used, in which the time to delay is provided (in milliseconds).

The next step was to write a web page for dashboard component. The page is written using "template" node, where the complete code of the page is provided and necessary data is being inserted. The page is written using HTML and Javascript along with Bootstrap 4 library.

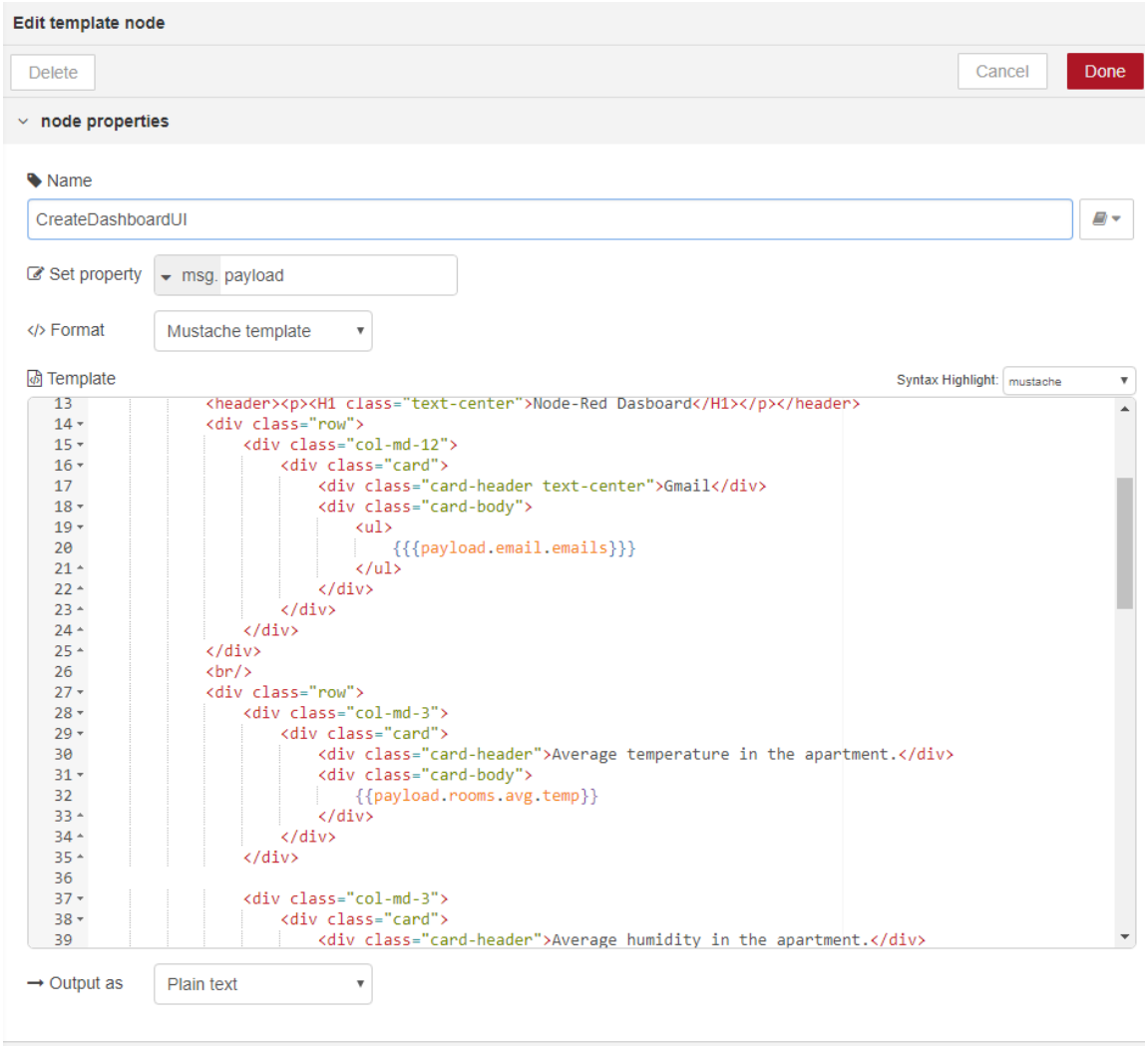
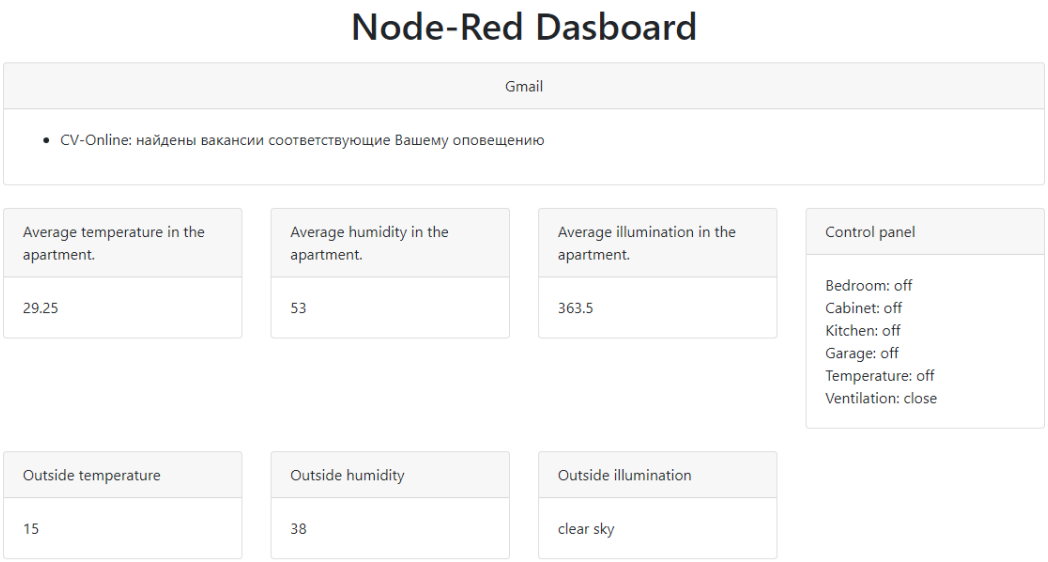


Figure 8 Content of the dashboard template node

In order to insert variables used in Nod-RED into HTML code, variable names should be enclosed in two curly brackets, as shown in the Figure 8.

Dashboard web page is depicted on the Figure 9.



© Copyright: German Novikov Student code:112587

Figure 9 Dashboard web page

In this flow the function that starts the work of entire simulation was written, the communication with mail and weather forecast services was embedded into the system as well. After Nod-RED receives and performs all the necessary operations, it creates a web page for displaying the values of sensors and commands that were sent to Smart Home management systems.

2.2.5 Simulation of Smart Home control system

In order to build a simulation of control systems of a Smart Home, an application that would accept commands from Nod-RED and validate incoming commands was created. This application is written in the Java programming language and installed on the Tomcat server.

In this application, there are three different classes for accepting commands and then performing corresponding validation of the commands accepted. For each class are provided its own validation rules and conditions. All classes accept data in JSON format

and then convert it to Java object using Jackson library. The Java objects are processed later on.

Detailed description of the Java classes that represent controller objects (each header below represents the exact name of the corresponding Java class):

- **LightController**

  This class is responsible for receiving commands to control the light. Commands are handled using MQTT protocol.

- **TemperatureController**

  This class is responsible for receiving commands to control the temperature. Commands are handled using HTTP protocol.

- **HumidityController**

  This class is responsible for receiving commands to control the humidity. Commands are handled using HTTP protocol.

2.3 Result of Built

As a result of developed simulation, different devices which use different communication protocols were successfully connected to each other and communication between them takes place with no known obstacles.

All devices are also associated with the application written in Node-RED. The Node-RED application describes how devices should interact with each other.

The general web page called dashboard was also successfully created during the development, using same Node-RED programming tool. All data that is being received from sensors, various services and data related to the status of management system during the simulation process is being displayed on the mentioned page.

3 Analysis

In this section, the work done is being analysed, considering the problems that occurred in the course of this work and providing explanation of applied solutions. Some possible future improvements are also considered in the given section.

3.1 Problems encountered in the course of work

In this section the problems encountered in the course of the work and their solutions are being discussed.

3.1.1 The problem with the response to the HTTP request

When writing requests through the HTTP protocol to obtain data from sensors, there was a problem that the sensors were unable to receive a response from the Node-RED application, thus exception in Java application occurred. This problem prevented the stable performance of the entire application.

After appropriate investigation it turned out, that the mentioned problem occurs if the "http" node is used alone, as it does not have the ability to send a response to the received request.

The solution to this problem was to use the "http response" node, in which the corresponding response was provided and then sent back to the client.

3.1.2 Concurrency problem with input data from two different nodes

If nodes are connected in such way that one node receives information from two different nodes, then in this situation arises concurrency problem: it is not possible to use the data from two nodes simultaneously. This problem was encountered when was needed to use the data coming from different nodes to build a dashboard.

To solve this problem, the node called "join" should be used. This note allows to create a new object, which contains the connected data that came from two different nodes. "join" node works as follows:

- By the specified path it takes the data from the received message object and puts it into the body of the created object.

- The name by which it is possible to find the data, the node takes from the path indicated by the user.

Thus, before combining the necessary information, the function that converts messages to the same data structure (message topic and content should be appropriately adjusted) has to be written.

3.1.3 In node “template” JavaScrip does not work with Node-RED variables

During writing a web page for dashboard, the following problem occurred: in the template code JavaScript was not able to get the current value of the Nod-RED variable.

This problem could be solved with the help of plug-in libraries for Nod-RED. Such libraries allow to use set of other (custom) nodes, including nodes which would solve the given problem. But due to the fact that the main aim of this thesis was to demonstrate that the standard Node-RED programming tool solution is sufficient for solving the stated problem and it is possible to write an application that allows to connect and control various devices of Smart Home system that communicate with each other using different protocols, without connecting third-party add-ons, no plug-in libraries were used during development.

In order to solve this problem, all JavaScript from the node "template" was moved to the separate "function" nodes, which became responsible for automatic generation of HTML code. This allowed to avoid the use of JavaScript in the node "template". However, taking into accordance the complexity of the structure and performance issues, the applied solution is not considered as the best way to solve this problem. It would be better to download a library that allows you to use the Node-RED variables in JavaScript without creating additional complex workarounds.

3.1.4 Delay in the operation of the circuit

The temporary delay in obtaining data occurred when data was loaded into the dashboard, therefore data on the page was displayed incorrectly (or was not displayed at all). The delay was caused by the fact that when someone requests (opens) the page, Node-RED sends a request to the simulation of sensors and control systems of the Smart Home to obtain corresponding data. Then the simulations in its turn sends requests to the Node-RED server, which processes received data and only then the results can be seen at the

dashboard. Therefore, it turned out, that the described problem in this case was related to the fact, that the dashboard requested data that was not yet known be the Node-RED server.

This problem was solved by putting a four-second delay before starting the simulation and creating a web page.

3.2 The work done, and how it could be improved

Achieved results

As the result of the carried out work a Smart House simulation was built. In the developed application various components were connected to the central program using different protocols. The application responsible for the communication flow between components was written using Node-RED programming tool. The program describes various scenarios of actions that can be performed depending on the defined sensor identifier (i.e. type). In the scenarios the commands sent to the Smart Home management systems are described.

Future improvements and remarks

Below are provided some possible improvements, which could be applied to the developed system (or similar system) in the future, along with related explanations and remarks as well.

1. Provided simulation was built using standard nodes provided by Node-RED tool. However, the application could be noticeably simplified, if third-party Node-RED libraries were used. Thus, for example, there is a library that provides ready solution for dashboard page [16], so separate dashboard HTML code and related JavaScript problems (see section 3.1.3) could be eliminated, custom nodes provided by the mentioned library could be used instead.

2. Also dashboard could be developed using Angular framework [17], as it allows to solve the problem of accessibility of the Node-RED variables in the JavaScript code in the “template” node (see section 3.1.3 for further information).

3. In this work not all if standard nodes provided by Node-RED programming tool were used, because there was no need to use other nodes, as it would not affect the provided solution results and thereby is out of scope of this work. However, other standard nodes can be easily added to the application, if needed.

4. In addition to the third position provided above, it is worth mentioning that if standard Node-RED nodes set does not provide needed node, which can fulfil stated requirements, then there is possibility to install third-party nodes. For that purpose, the library containing such nodes is provided on the official Node-RED web page [18].

5. If there is no appropriate node in the list of standard Node-RED nodes nor in the provided Node-RED library, Node-RED allows to create your own custom node which can be then integrated with any existing application.

4 Summary

The aim of this thesis was to find a solution to the problem of creating Smart Home system, which consists of the devices that use different communication protocols, without the need of creating a complex program or being restricted in the choice of components, i.e. using components from one provider only. In the given work Node-RED programming tool is proposed as the solution of the stated problem.

In the course of the work a simulation of the Smart Home system was built using Node-RED programming tool. Simulation of the sensors behavior was written for four rooms, each of which has four different sensors: 1) temperature measuring sensor, 2) humidity sensor, 3) illumination sensor 4) and switch state sensor, which handles the state of the light switch (i.e. simulated physical device located in the room, which can be in “ON” or “OFF” state). Each room sends its sensors data to the Node-RED server using different data standards and communication protocols, where running Node-RED application receives the requests, analyses obtained data and after that in its turn sends appropriate commands to the Smart Home control systems.

Carried out work has shown how Node-RED can be used for solving the problem of implementing Smart Home system using together components, which support different communication protocols. Given programming tool allowed to build Smart Home simulation where components are successfully communicating with each other using different protocols without the need of implementing complex expensive program.

References

[1] “Node-RED” [Online]. Available: https://nodered.org/ [Accessed 1 May 2018]

[2] “Wikipedia. HTML” [Online]. Available: https://en.wikipedia.org/wiki/HTML [Accessed 19 May 2018]

[3] “MQTT” [Online]. Available: http://mqtt.org/ [Accessed 19 May 2018]

[4] “XML.com A Technical Introduction to XML” [Online]. Available: https://www.xml.com/pub/a/98/10/guide0.html#AEN58 [Accessed 19 May 2018]

[5] “Introducing JSON” [Online]. Available: https://www.json.org/ [Accessed 19 May 2018]

[6] “Wikipedia. Uniform Resource Identifier” [Online]. Available: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier [Accessed 1 May 2018]

[7] “Oracle. What Is a URL?” [Online]. Available: https://docs.oracle.com/javase/tutorial/networking/urls/definition.html [Accessed 1 May 2018]

[8] “Webopedia. HTTP - HyperText Transfer Protocol” [Online]. Available: https://www.webopedia.com/TERM/H/HTTP.html [Accessed 5 May 2018]

[9] “Electronics Research Group. Transmission Control Protocol (TCP)” [Online]. Available: http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/tcp.html [Accessed 5 May 2018]

[10] “Mozilla. UDP (User Datagram Protocol)” [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/UDP [Accessed 18 May 2018]

[11] “Wikipedia. Java (programming language)” [Online]. Available: https://en.wikipedia.org/wiki/Java_(programming_language) [Accessed 19 May 2018]

[12] “Apache Tomcat” [Online]. Available: http://tomcat.apache.org/ [Accessed 1 February 2018]

[13] “Node.js” [Online]. Available: https://nodejs.org/en/ [Accessed 1 February 2018]

[14] “Eclipse Mosquitto” [Online]. Available: https://mosquitto.org/ [Accessed 29 April 2018]

[15] “HTTP Status Codes” [Online]. Available: https://httpstatuses.com/ [Accessed 1 May 2018]

[16] “GitHub node-red-dashboard” [Online]. Available: https://github.com/node-red/node-red-dashboard [Accessed 15 May 2018]

[17] “Angular” [Online]. Available: https://angular.io/ [Accessed 16 May 2018]

[18] “Node-RED. Node-RED Library” [Online]. Available: https://flows.nodered.org/?num_pages=1 [Accessed 19 May 2018]

Appendix 1 – Java class of BedroomSensorsContainer

```
public class BedroomSensorsContainer implements Room {

    private static  boolean statusOfSwitch = false;

    public void sendTemperature() {
        StateComponent temperature = new
StateComponent(Sensors.TEMPERATURE.toString(),
                String.valueOf(getRandomValue(Room.minTemperature,
Room.maxTemperature)));
        createConnection(Sensors.TEMPERATURE.toString().toLowerCase(),

MessagesHelper.getComponentAsJSONStrting(temperature));
    }

    public void sendHumidity() {
        StateComponent humidity = new
StateComponent(Sensors.HUMIDITY.toString(),
                String.valueOf(getRandomValue(Room.minHumidity,
Room.maxHumidity)));
        createConnection(Sensors.HUMIDITY.toString().toLowerCase(),
                MessagesHelper.getComponentAsJSONStrting(humidity));
    }
    public void sendIllumination() {
        StateComponent illumination = new
StateComponent(Sensors.ILLUMINATION.toString(),
                String.valueOf(getRandomValue(Room.minIllumination,
Room.maxIllumination)));

createConnection(Sensors.ILLUMINATION.toString().toLowerCase(),

MessagesHelper.getComponentAsJSONStrting(illumination));
    }
    public void sendStatusOfSwitch() {
        statusOfSwitch = !statusOfSwitch;
        StateComponent switchMessage = new
StateComponent(Sensors.SWITCH.toString(),
                String.valueOf(statusOfSwitch));
        createConnection(Sensors.SWITCH.toString().toLowerCase(),

MessagesHelper.getComponentAsJSONStrting(switchMessage));
    }

    public int getRandomValue(int from, int to) {
        Random rand = new Random();

        int  result = rand.nextInt(to) + from;
        return result;
    }

    private void createConnection (String to, String message){
        String address = "http://127.0.0.1:1880/bedroomHandler/" + to;

        HttpClient httpClient = HttpClientBuilder.create().build();
        try{
            String payload = message;
            StringEntity entity = new StringEntity(payload,
                    ContentType.APPLICATION_JSON);
```

```
            HttpPost request = new HttpPost(address);
            request.setEntity(entity);
            HttpResponse response = httpClient.execute(request);
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```