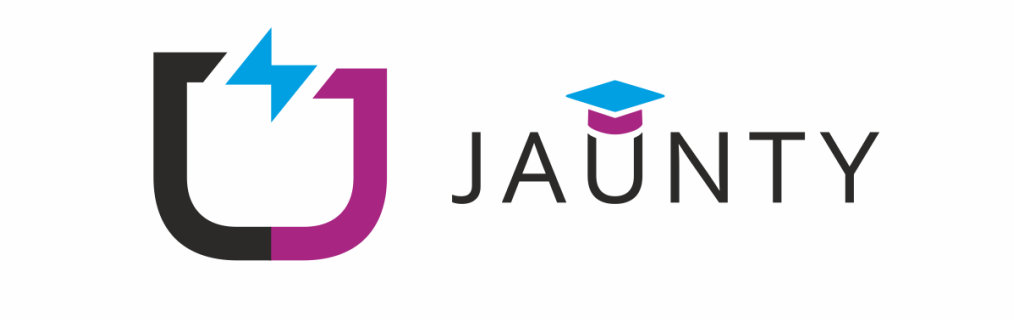
IoT workflows with Node-RED



Author: Public Power Corporation S.A

Co-funded by the
Erasmus+ Programme
of the European Union

Copyright

Consisting of

Coordinator:	Technical University of Sofia	Bulgaria
Partners:	University of Western Macedonia	Greece
	International Hellenic University	Greece
	Public Power Corporation S.A.	Greece
	University of Cyprus	Cyprus
	K3Y Ltd	Bulgaria
	Software Company EOOD	Bulgaria

Co-funded by the Erasmus+ Programme of the European Union

Table of Contents

1. Abbreviations 3
2. Scope 4
   2.1 Specific outcomes 4
   2.2 General description 4
   2.3 Lab configuration 6
3. Exercise 1: Introduction 7
   Step 1: Hello World! 7
   Step 2: The function node 10
   Step 3: The switch node 10
4. Exercise 2: Simple Dashboard 14
   Step 1: Setup 14
   Step 2: Collect and show the Measurements 17
   Step 3: The range node 19
5. Exercise 3: Control 20
   Step 1: Frequency 20
   Step 2: On and Off 26
   Step 3: Presentation 27
6. Appendix I: NodeRED project 33
7. Appendix II: Circuit sketch and source code 36
8. References 40
9. Contacts 41

1. Abbreviations

GUI	Graphical User Interface
UI	User Interface
IoT	Internet of Things
MQTT	MQ Telemetry Transport
JSON	JavaScript Object Notation

2. Scope

The scope of this laboratory exercise is to introduce the Node-RED tool as a tool to design and implement a complete IoT solution. In this regard, we will describe a specific use case, investigate the design decisions one could make and implement the final solution using Node-RED.

2.1 Specific outcomes

Upon completion of this lab, individuals will be able to:

- Understand and use the Node-RED platform.
- Receive Measurements and create a dashboard, using Node-RED.

2.2 General description

In this laboratory, we will use a circuit comprising of a load and a distance sensor.

For the purposes of the lab, we can imagine that the circuit used will be part of a Smart Bin, in the context of a smart waste management system of a Smart City [1]. More specifically the circuit will include a load sensor and a distance sensor. The purpose of this lab is to receive these measurements and present them in a dashboard created in Node-RED [2].

The sensors that will be used are:

- A common Load Cell/ Strain Gauge.
- The HX711 - Load Cell Amplifier.
- The Ultrasonic ranging module HC - SR0X.

The load cell, shown in Figure 1, is a force transducer that translates the straining force applied to it to electrical voltage, using a Wheatstone bridge [3]. The two ends of the load cell will usually be mounted to two different plates. The bottom plate would be steady, while the upper plate would move vertically based on the placed weight. This movement causes strain in the middle, flexible part of the load cell, which produces electrical voltage. Depending on the scale of weights that we want to be measured, there are different load cells. Ranging from some grams to kilograms. Of course, as the weight increases, the granularity decreases.

Figure 1: Load Cell

To translate this voltage to a useful signal, we will use the HX711 Load Amplifier (shown in Figure 2) that translates the raw voltage changes to digital signals. Many Arduino libraries have been written for the HX711, which makes the connection and measurement retrieval easy [4].

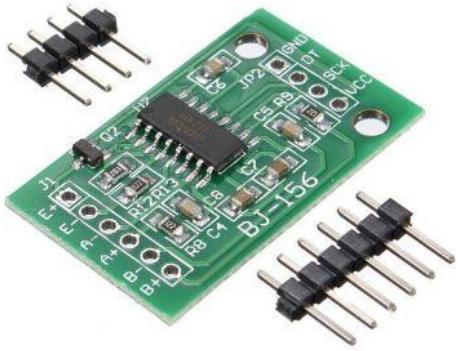


Figure 2: HX711 Load Cell Amplifier

Finally, the HC-SR04 Ultrasonic Sensor will produce distance measurements [5]. HC-SR04 operates by sending a sound wave at a frequency too high for a human to perceive and calculating the time it takes it to return to its source. Usually, it consists of a separate emitter and receiver as shown in Figure 3. As with the load cell, the HC-SR0X family consists of many different sensors depending on the maximum distance we want to measure. The X in the name denotes the maximum meters measured. Of course, as the maximum distance increases, granularity decreases.

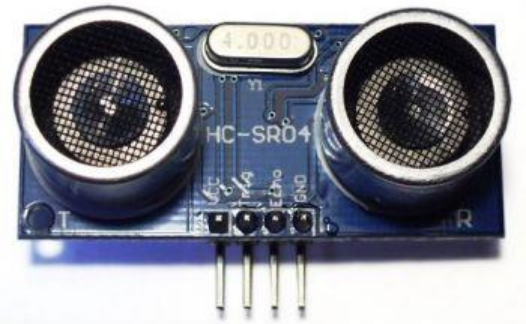


Figure 3: HC-SR04 Ultrasonic sensor

The circuit with the sensors has been deployed and is running in the lab. The measurements are sent to specific MQTT topics. More specifically the measurements from HC-SR04 are sent to the "Bin/HCSR04" topic and the measurements from the load cell are sent to the "Bin/HX711" topic.

If you want to create the circuit yourselves, schematics and code are provided in Appendix II.

2.3 Lab configuration

The following parameters are provided by the lab instructor:

Property	Value
Workstation IP address	
Workstation username	
Workstation password	
Access method for the workstation	VNC
MQTT Broker	

3. Exercise 1: Introduction

Step 1: Hello World!

The Node-RED platform is installed by default in your workstation. To start with, login to your machine (check the information of section 2.3), and start the platform by issuing the following command on the terminal:

```
node-red
```

After node-red initializes you may access the web-gui through any web browser in:

```
http://localhost:1880/
```

The Web GUI should look something like Figure 4.

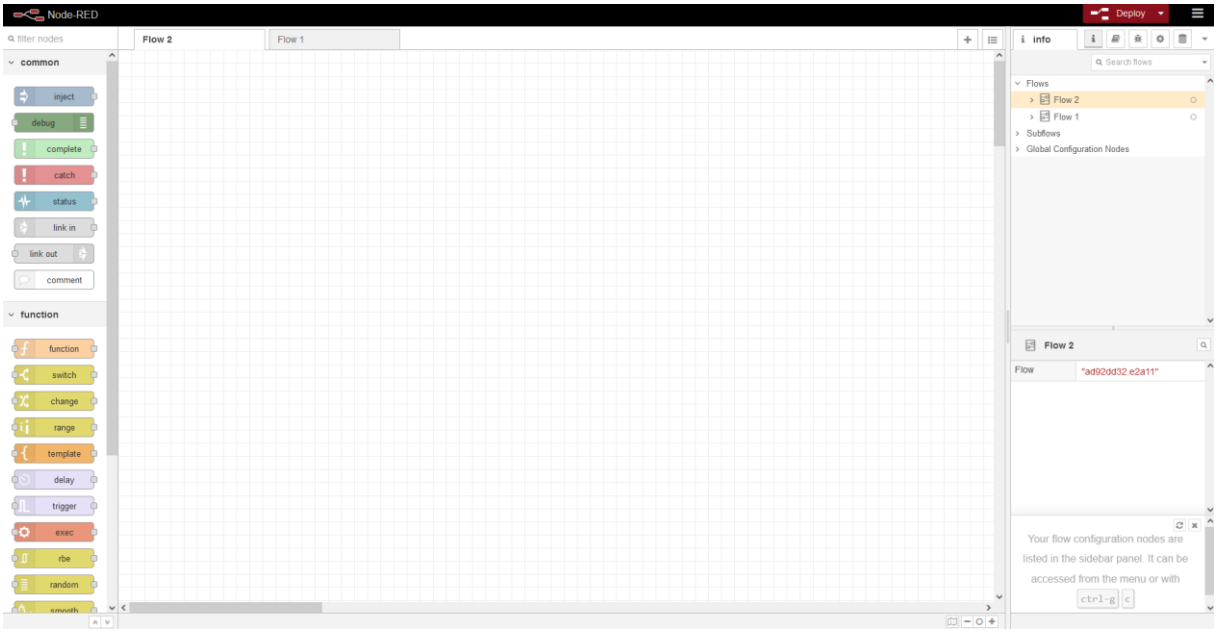


Figure 4: Preview of the Node-RED homepage

On the centre of the screen is the main canvas, where we will build our flows. On the left side, there are the various available components and on the right side, various information is presented based on what we are working on.

Alternatively, you could install Node-RED to your local PC, or you could use FRED: Front End for Node-RED, which is a cloud hosted instance of Node-RED. FRED is subscription based, but has a free option, with limitations.

Let's go on to create our first Flow! For this introduction we will use the inject node and a debug node, which should be the first two nodes in the list on the left, under **Common**. Drag these two components on the canvas and then connect them by clicking and dragging on the grey square on the right of the

inject node to the grey square on the left of the debug node. This grey square is called “port” and the process of connecting different nodes through ports is called wiring. Your flow should look like Figure 5.

Figure 5: The first Node-RED flow

Now by double clicking on the timestamp node, a panel should appear on the right as shown in Figure 6.

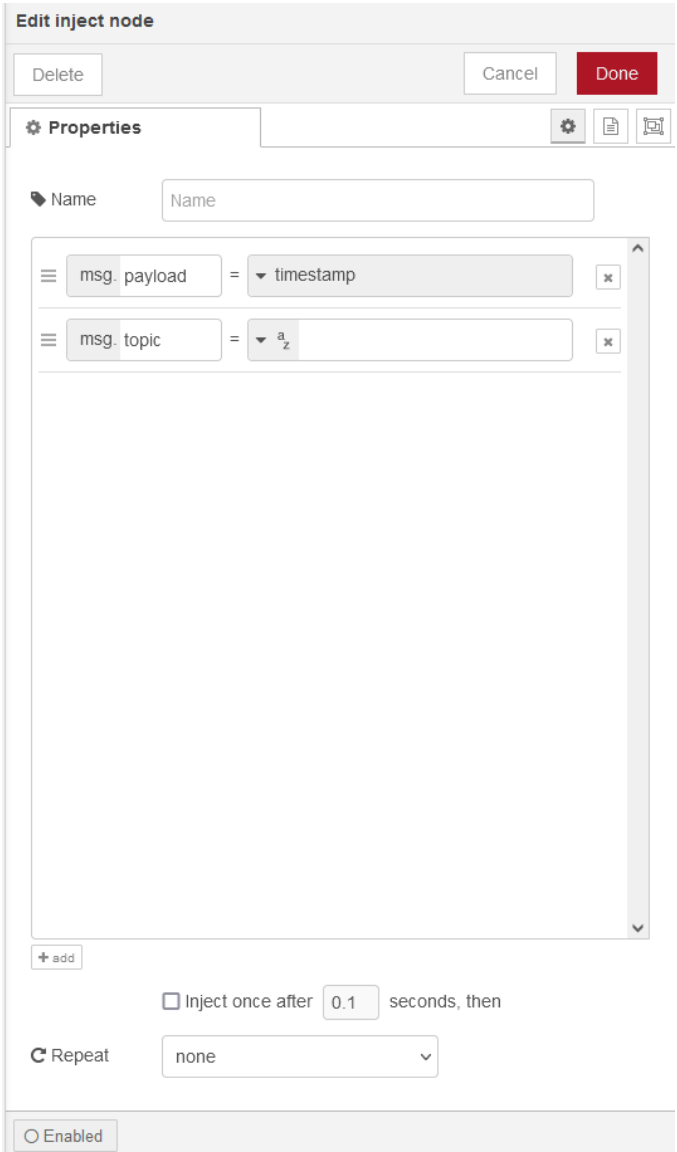


Figure 6: Editing the timestamp node

Here you can set the various parameters of the inject node. For now, we will change the field that is now “timestamp” to “string” and add “Hello World!”, as shown in Figure 7.

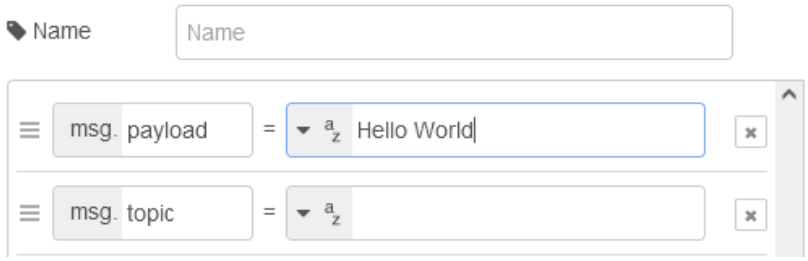


Figure 7: Editing the values that the timestamp node injects

Next double click on the Debug node. A similar panel should appear. Make sure that that it set as shown in Figure 8.

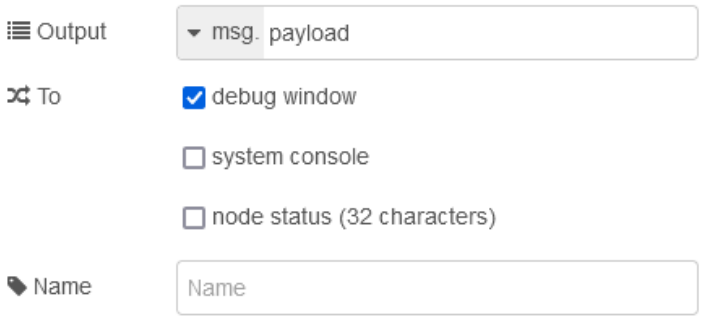


Figure 8: Editing the debug node

We are almost ready! Hit the Deploy Button on the top right and wait for the “Successfully deployed” message.

Now by clicking the button on the left of the inject node, you should receive a message saying, “Successfully Injected Hello World!”

Also, by selecting the debug tab in the rightmost panel, you will see the message printed by the debug node, depicted in Figure 9.

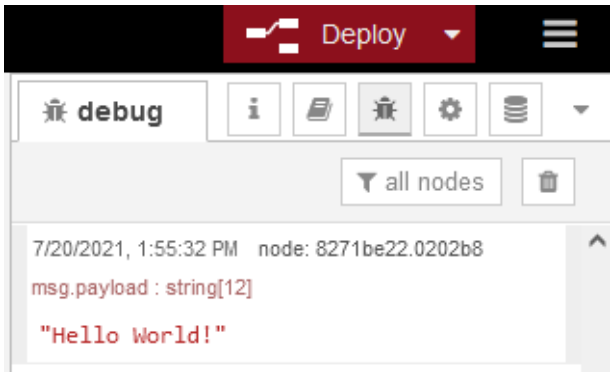


Figure 9: The output of the debug node

Step 2: The function node

Let's now change a little the inject node. Double click again and change the msg.payload to "timestamp" as it originally was. Hit the Deploy button and try again to see what it shows.

The message you will see will consist of some numbers that do not make sense by themselves. We have to find a way to convert this number to a human readable format.

For such things we use the function node that you can find under the Function list on the leftmost panel. Drag a function node to the canvas. Remove the wire that was used to connect the inject and the debug node and connect the inject output to the function input and then the function output to the debug input. The new flow is depicted in Figure 10.

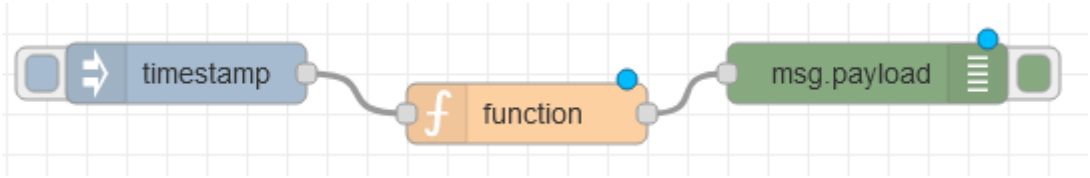


Figure 10: The new interconnection including the function node

By double-clicking the function node, you will be presented with an editor. Here you can write your own custom JavaScript code to be used. Make sure you are on the On Message tab and insert the following code:

```
1  msg.payload = new Date(msg.payload);
2  msg.hours = msg.payload.getHours();
3  msg.minutes = msg.payload.getMinutes();
4  return msg;
```

The code above converts the numbers of the incoming message to a proper Date object in line 1. In line 2 and 3 we keep the specific values of hours and minutes (this will be used in the next step). Then on line 4 it publishes the modified message to the output port.

Now If you hit the inject button you should see the current date and time.

Don't forget to hit the Deploy button before testing some new feature. Node-RED will insert a blue circle on the top right of every node that has changes since last deployment.

Step 3: The switch node

Another important node that we will be using is the switch node. You can find the switch node under the function list on the leftmost panel.

Drag the switch node to the canvas and double click it. Here we select which property we want to compare to what case. In essence this is a switch statement like in C or any other language. By clicking

the “add” button under the case table you can add more cases, which will correspond to different output ports. In each case you can select which operator to apply and what will the operand be. Additionally in the Property field you can select which field of the msg you want to compare to.

First of all, let’s select the Property field to be msg.hours. Then insert a new rule, so that we have two cases. In the first case we will compare msg.hours > 12, in the second case msg.hours <= 12. The switch node should look like this.

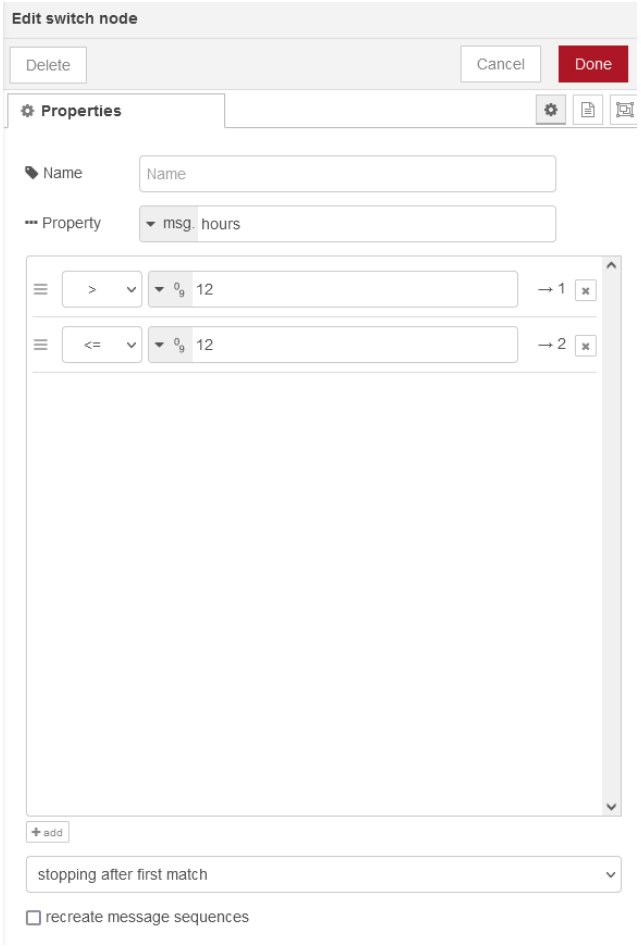


Figure 11: Editing the switch node

Back to the canvas you will see that the switch node now has two output ports. Connect them to new different function nodes as shown in the image below:

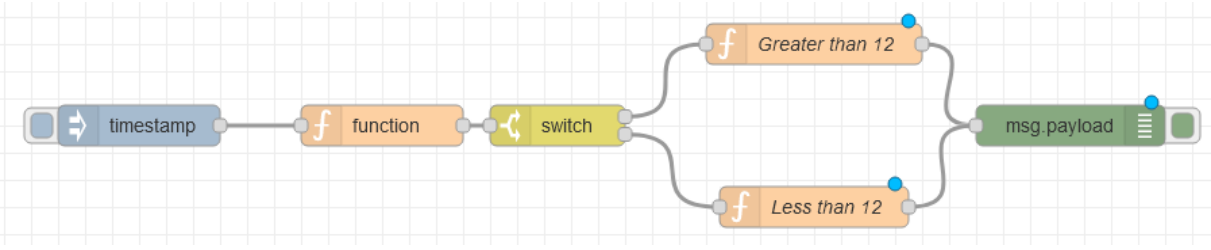


Figure 12: Updated flow including the switch node

We have changed the Name property of the new function nodes to make them easier identifiable. The "Greater than 12" node will be active if the switch node fires port number 1, which will happen if msg.hours > 12. Respectively for the "Less than 12" node.

The "Greater than 12" node should have this code:

```
1  msg.payload = "It's " + msg.hours + ":" + msg.minutes + " pm. Good Evening!";
2  return msg;
```

And the "Lesser than 12" this one:

```
1  msg.payload = "It's " + msg.hours + ":" + msg.minutes + " am. Good Morning!";
2  return msg;
```

What that code does is to create a string form the hours and minutes that we had extracted in Step 2.

Now deploy the flow and try it. You should see different outcomes depending on what time it is.

Take your time to familiarize yourself with the Node-RED GUI before continuing. Try changing different parameters in the various nodes and the wiring to see how the functionalities change. You are encouraged to do this after every step for the rest of the Lab, to better grasp what we have done in each step.

Before going forward it is useful to know that Node-RED has a good built-in documentation of the various nodes. You can access it by selecting the help tab in the rightmost panel next to the debug tab, as shown in Figure 13.

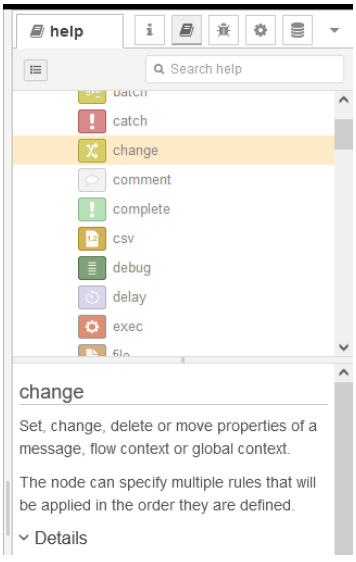


Figure 13: The help tab of Node-RED

Finally, you can export your flows to JSON format, to share your work with other people. To do this you should click on the menu on the top right and select export. On the menu that will be presented, make sure that you have selected all the nodes of the current flow. The export dialogue is depicted in Figure 14.

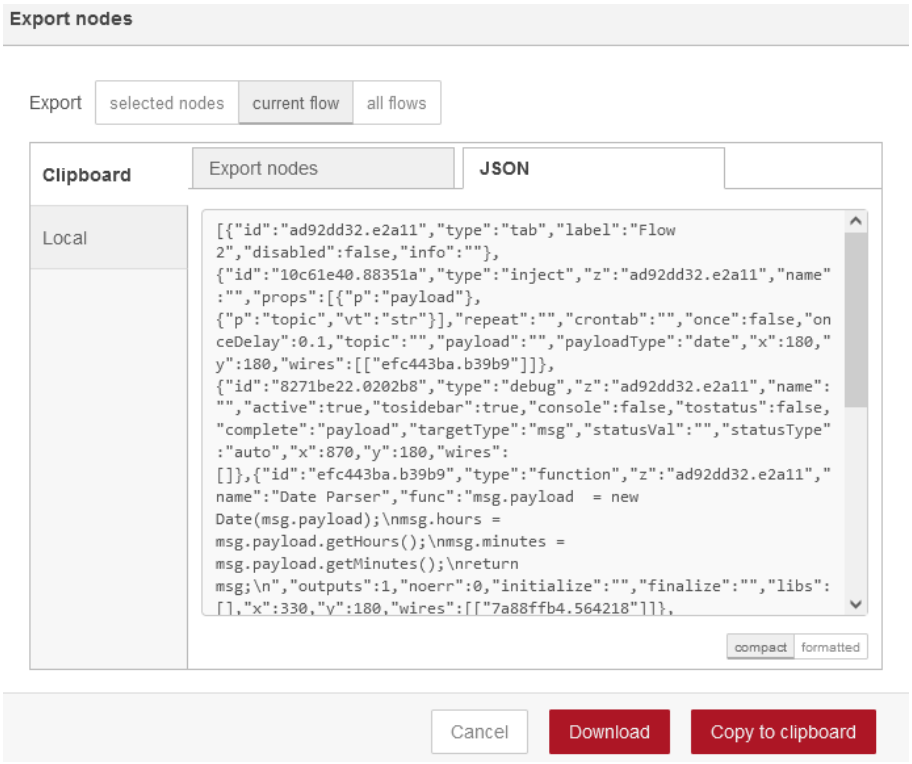


Figure 14: Exporting the flow in JSON format

By selecting import you can also load other people's flows. Try it now with the code below, which represents the last example we have built.

```
[{"id":"ad92dd32.e2a11","type":"tab","label":"Flow
2","disabled":false,"info":""},{"id":"10c61e40.88351a","type":"inject"
,"z":"ad92dd32.e2a11","name":"","props":[{"p":"payload"},{"p":"topic",
"vt":"str"}],"repeat":"","crontab":"","once":false,"onceDelay":0.1,"to
pic":"","payload":"","payloadType":"date","x":180,"y":180,"wires":[["e
fc443ba.b39b9"]]},{"id":"8271be22.0202b8","type":"debug","z":"ad92dd32
.e2a11","name":"","active":true,"tosidebar":true,"console":false,"tost
atus":false,"complete":"payload","targetType":"msg","statusVal":"","st
atusType":"auto","x":870,"y":180,"wires":[]},{"id":"efc443ba.b39b9","t
ype":"function","z":"ad92dd32.e2a11","name":"Date
Parser","func":"msg.payload  = new Date(msg.payload);\nmsg.hours =
msg.payload.getHours();\nmsg.minutes =
msg.payload.getMinutes();\nreturn
msg;\n","outputs":1,"noerr":0,"initialize":"","finalize":"","libs":[],
"x":330,"y":180,"wires":[["7a88ffb4.564218"]]},{"id":"7a88ffb4.564218"
,"type":"switch","z":"ad92dd32.e2a11","name":"","property":"hours","pr
opertyType":"msg","rules":[{"t":"gt","v":"12","vt":"num"},{"t":"lte","
v":"12","vt":"num"}],"checkall":"false","repair":false,"outputs":2,"x"
:490,"y":180,"wires":[["48378fb4.560ef8"],["9baf475.8489f38"]]},{"id":
```

```
"48378fb4.560ef8","type":"function","z":"ad92dd32.e2a11","name":"Great
er than 12","func":"msg.payload = \"It's \" + msg.hours + \":\" +
msg.minutes + \" pm. Good Evening!\"\nreturn
msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","libs":[],"x
":680,"y":120,"wires":[["8271be22.0202b8"]]},{"id":"9baf475.8489f38","
type":"function","z":"ad92dd32.e2a11","name":"Less than
12","func":"msg.payload = \"It's \" + msg.hours + \":\" + msg.minutes
+ \" am. Good Morning!\"\nreturn
msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","libs":[],"x
":680,"y":240,"wires":[["8271be22.0202b8"]]}]
```

4. Exercise 2: Simple Dashboard

Step 1: Setup

Now that we have seen the basic nodes and features of Node-RED, we can move on to create a simple dashboard for our IoT application. To begin we will need to install a non-standard module. Modules are like libraries of a programming language. They provide ready nodes that implement specific function and can be used in our flows.

You can install modules easily through the Node-RED frontend. In the top right menu select Settings. Then on the Palette tab, you will see the installed modules and the available modules to be installed. Select the Install tab as seen on Figure 15 then search for “node-red-dashboard” to install.

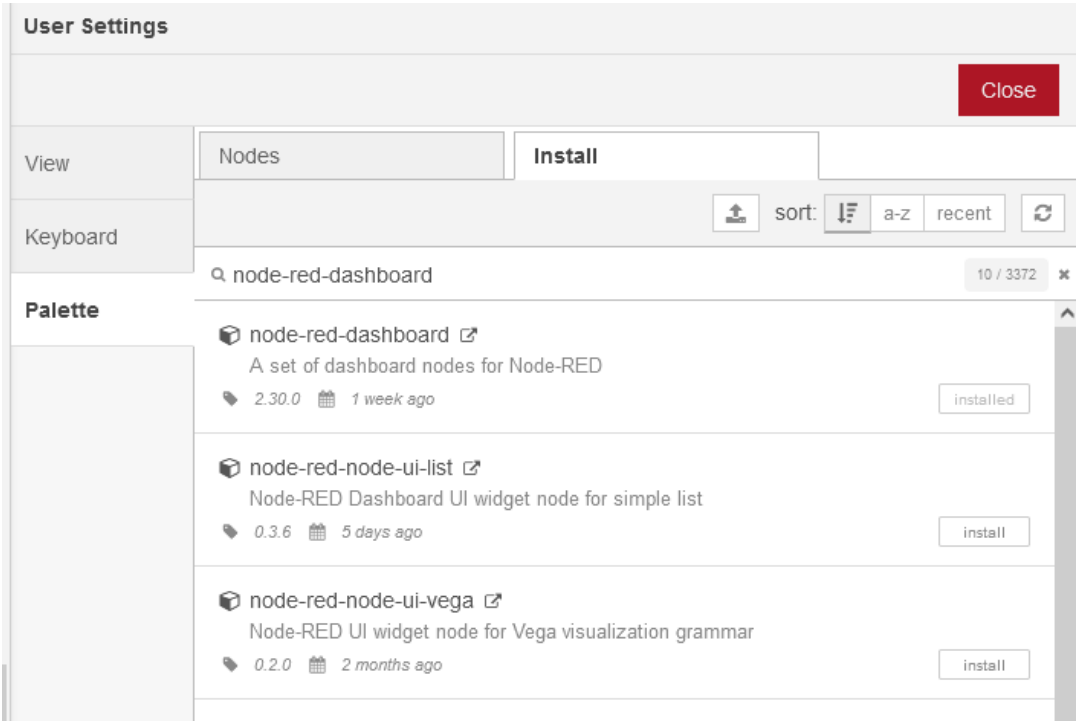


Figure 15: Installing the node-red-dashboard addon

After you have installed the node-red-dashboard module, you should have a new tab in the right-hand menu, called dashboard, which will be empty. Here you can set the various details for the dashboard you will create. On the Site tab, you can set the parameters that concern the Site itself, like its name, the presentation etc. On the Theme tab, you can change the way your dashboard will look concerning colours etc.

The most important part, however, is the **Layout** tab, depicted in Figure 16. Here you can create and set how the various nodes will be placed and displayed on the dashboard. The two entities that exist are “Tabs” and “Groups”. A Page can have many Tabs and each Tab can have many Groups. When you initialize a node-red-dashboard node, you will have to configure to which Group that node will belong, which will in turn dictate where it will be rendered. For now, just create a Tab and two Groups by clicking the appropriate buttons. Then you can edit each Tab or Group to change their names. For now, let’s keep the default names.

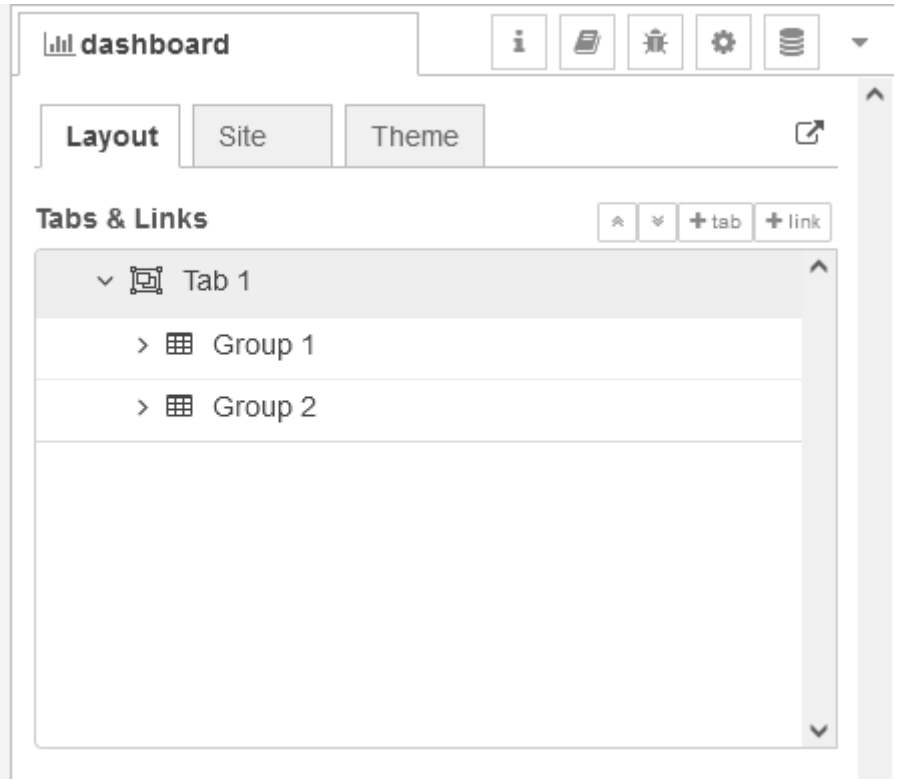


Figure 16: The layout tab of a dashboard

Next, add a text node and a switch node from the dashboard group and connect them.

Different nodes can have the same names. For example, switch node from the function and the dashboard group. Be careful when following the steps to select the right nodes.

Now by double clicking on each node, you will assign the switch to Group 1 and the text to Group 2. Also change the label of each node to “This is in Group X”, where X is the appropriate group.

You should have the flow depicted in Figure 17.

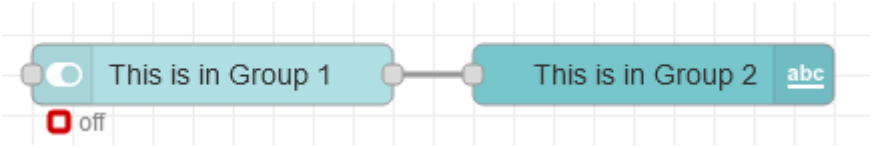


Figure 17: A flow of groups

In this flow, the value of the switch object will be printed as text on the text object.

Deploy and access the dashboard at

```
http://localhost:1880/ui/
```

You should see something like Figure 18.

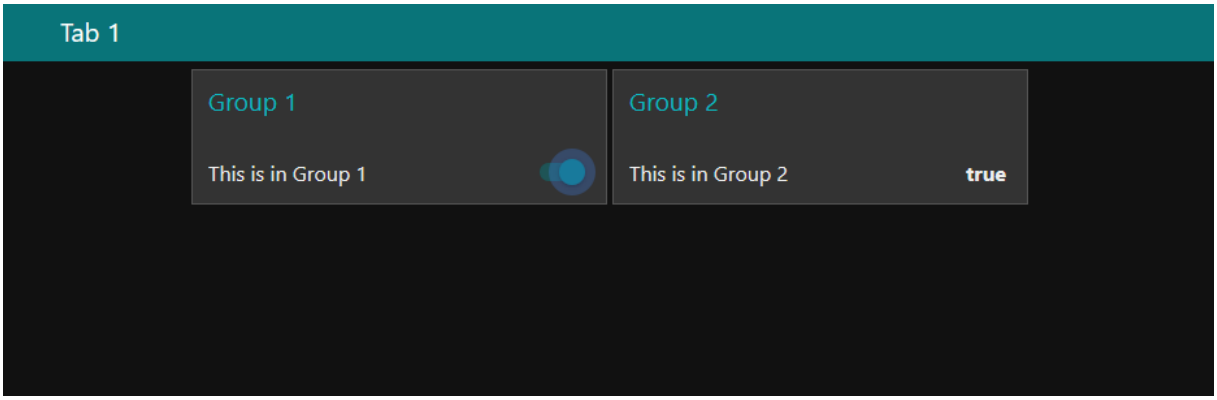


Figure 18: The new Node-RED dashboard

Note that Groups are not closed boxes, you can still pass values from one Group to the other, they are simply a way to set the layout.

If you close the dashboard and reopen it, you will see that the switch is where you left it. That is because the dashboard is running on the Node-RED regardless of if you are actively accessing it. If you want to set the switch to a default value, you can use the UI control node in the dashboard section.

The UI control node acts like the inject node of exercise 1, but it fires when a certain event happens. By double clicking it, you can set it to fire on the specific event you want, by changing the Output field. Set it now to “Connect event only”. That means that the UI control node will fire when a new connection is made to the dashboard.

Then by using a function node, we will set the msg.payload to the false value.

```
1  msg.payload = false;
2  return msg;
```

The function node will take as input the output of the UI control and will output to the input of the switch node. When something is received by the switch node, it will change its value. The new flow is depicted in Figure 19.

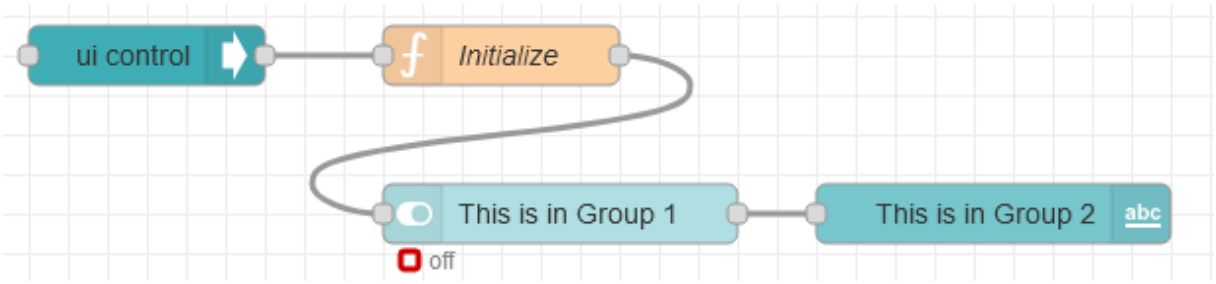


Figure 19: The updated flow with ui control

Now every time you access the dashboard, the value of the switch should be false.

Step 2: Collect and show the Measurements

Now that we have grasped the basic components of a dashboard let's move on to creating a real dashboard to show the measurements we receive from the sensors. As we have described in the beginning, we will be using two sensors that send measurements using MQTT. The two topics we should monitor are "Bin/HCSR04" and "Bin/HX711".

In Node-RED, there exist specific nodes that can subscribe and publish to MQTT topics. You will find them under the network group as mqtt-in and mqtt-out. By double clicking them, you can set the various parameters, like which server you want to connect to, which topic to publish/subscribe, set QoS etc.

In our case, we will need two mqtt-in nodes, connecting to the localhost server on port 1883. One will subscribe to the Bin/HCSR04, the other to the Bin/HX711. To test if everything is all right, we will just send the message receive to two debug nodes, one for the payload and one for the topic. This scheme is depicted in Figure 20:

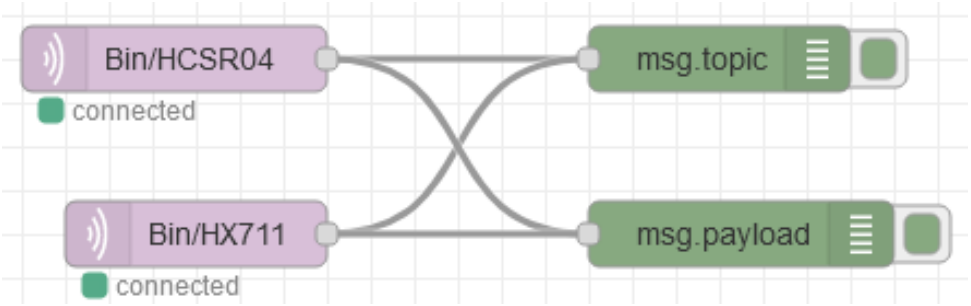


Figure 20: Connecting MQTT subscribers with debug nodes

We could use only one mqtt-in node to subscribe to the topic Bin/#, and then run a switch node on the msg.topic for each different topic that we want to do something different. This would not alter the efficiency of our program but would complicate the high-level view. Try implementing it as an exercise!

So, after making sure that we can connect to the MQTT server and receive the measurements, it is time to finally show them on a simple dashboard. There are two nodes that we can use as a basis. The gauge node and the chart node, both under the dashboard group.

Each of these nodes has some customization options that will alter the way the measurements are represented. In our exercise, we will use a gauge to present the measurements from the HCSR04 and a chart for the HX711. These elements belong to the dashboard module so be sure to define in which Group they belong, i.e., where they will be placed on the dashboard. Feel free to change the previous names to something more specific like Load, Distance etc, or even remove and add Groups and Tabs. The flow should look like Figure 21.

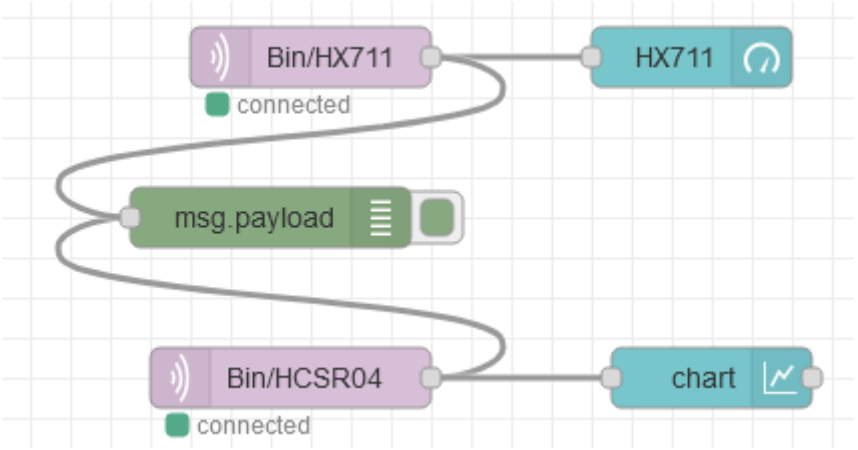


Figure 21: Connecting the MQTT clients with the dashboard

The Dashboard should look something like Figure 22.

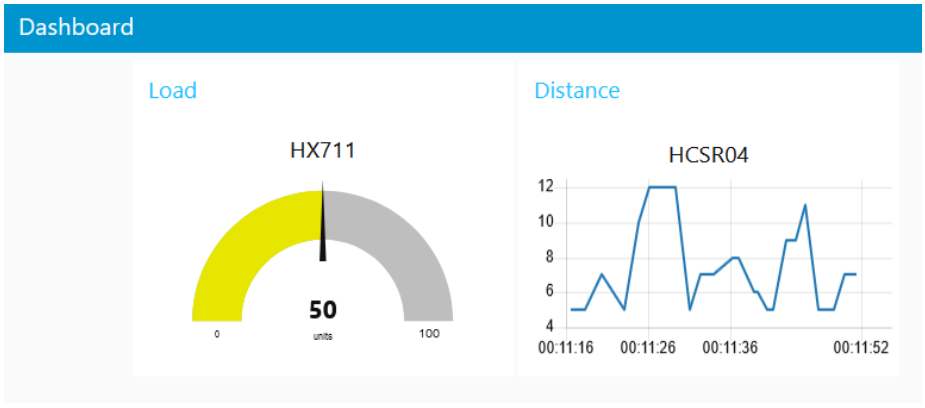


Figure 22: The dashboard visualising the measurements

Step 3: The range node

You may have noticed that the measurements you are receiving are not normal, i.e., they do not seem to correspond to logical values in the range of what is meant to be measured. That is because the sensors are not calibrated. What you receive are the raw values the sensor is giving. This should be translated to the corresponding logical values. Usually this would be done in the sensor side, but for this scenario let's see how we can easily implement such a transformation.

To achieve the calibration, we need to know if the transformation is linear, or it follows some specific rule. In our case both sensors have a linear reading. Which means that we need just two definitive measurements, i.e., two points that we already know what they translate to. This could be for example a measurement of something that we know is one kilogram, or the measurement of a specific distance like 1 meter.

Then we can use the range node to perform the transformation. You will find the range node under the function group. By double clicking it you must enter 4 integers (Figure 23). The input range should be the two measurements we have received, and the target range should be the definitive measurement we know that it corresponds to.

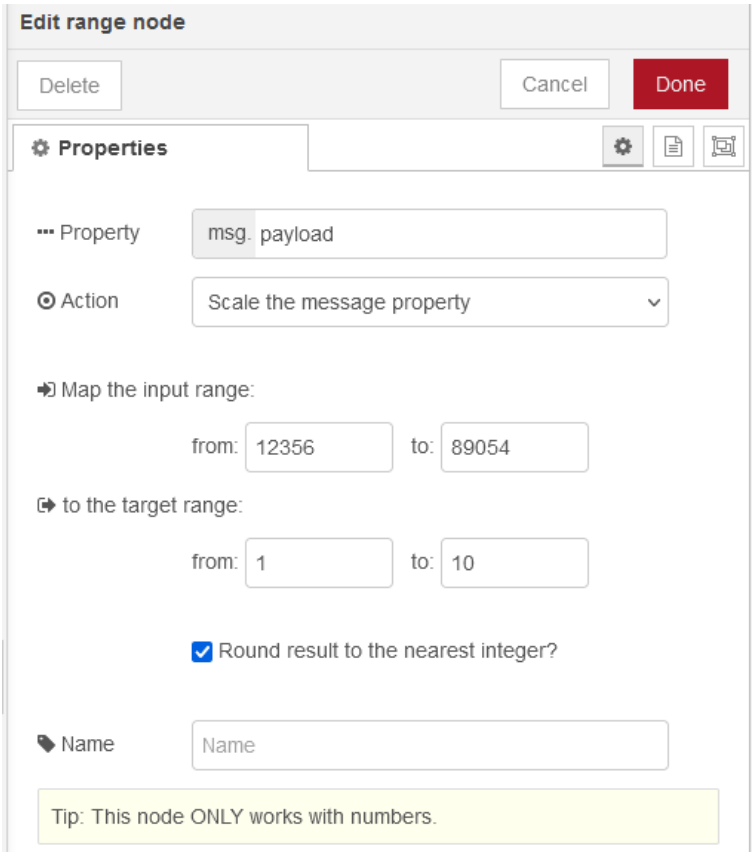


Figure 23: Editing the range node

Insert two such configured range nodes between the mqtt-in node and the gauge/chart node. The measurements that are displayed should now be logical.

5. Exercise 3: Control

Step 1: Frequency

Now, the gateway sends measurements is a predefined frequency. However, there is implemented a function to change the sending frequency of the measurements. The gateway subscribes to the topic Bin/freq. If a number is received in that topic, the gateway will change the frequency (in seconds) in which it sends measurements.

To implement this on the dashboard we will need some way of input for the user to select the frequency and then we should send this number to the Bin/freq topic. For the input we will use the text input node of the dashboard module and to publish the user input we will use an mqtt out node of the network list. Also add a debug node to the output of the text input node, like Figure 24.

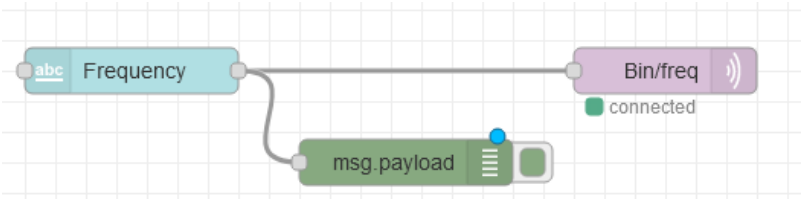


Figure 24: Updating dashboard with frequency

The mqtt-out node has almost the same configuration with the mqtt-in nodes. We set the broker address and the topic we want to publish to. The broker address is the same as in the previous step and the topic is Bin/Freq. Make sure that you also set the retain flag to True. By double clicking the text input node, you should open a menu like Figure 25.

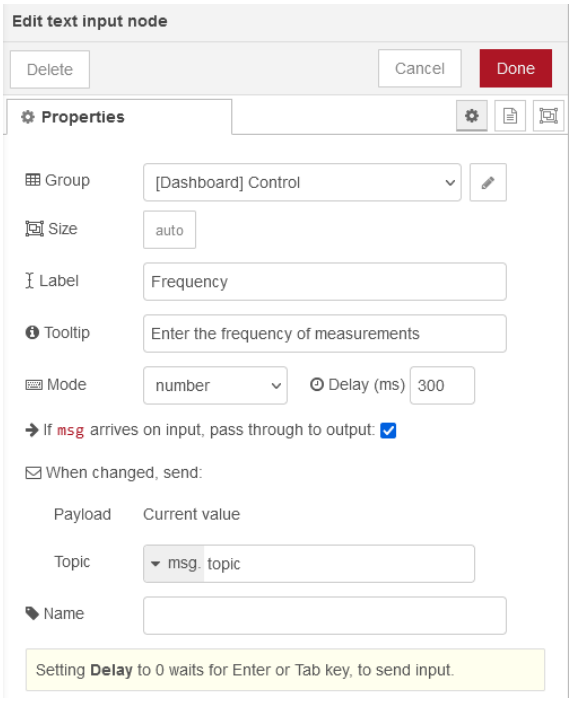


Figure 25: Edit the text input node

First, set the Group that this node belongs to a new Group named Control. Then on the Label field, we set what the title of the input will be. In our case we set it to frequency. An optional tooltip could also be set to provide a description. On the Mode field, we can set what the input will be, which also changes the form of field the user sees. For our case we will set it to number.

An important field is the delay field. Here we set the number of msec, after which the input will be sent forward. For example, the default value is 300ms. Meaning that after the user stops writing something for 300ms, the value of the field will be forwarded. If we set it to zero, the value will not be sent unless the user presses the Enter or the Tab key. For now, we will leave it at 300ms.

Deploy the flow and you should have a dashboard like in Figure 26.

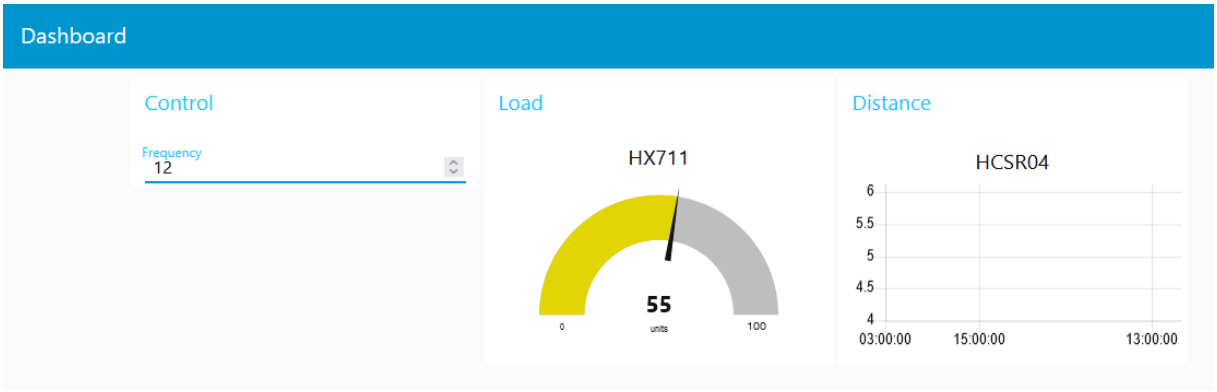


Figure 26: The updated dashboard with frequency control

Now try changing the value of the text input and you should see in the debug view that with every change, a message is fired. That message is also published to the MQTT broker. This is bad design because the MQTT broker will be flooded with useless messages as we change the frequency. We want to have a way to send the message only when we are sure about the frequency. One way as we described before is to set the delay of the input node to 0ms. Try it now.

You should notice that while it does not flood with messages as previously, now it sends only when we press the Enter key and not when changing the numbers with just the up/down keys. That behaviour is not intuitive.

Set again the delay of the input node to 300ms.

We would like for the user to have a definitive way to send the frequency. For that we will use a button node from the dashboard module.

Double clicking on the button node should bring the menu depicted in Figure 27.

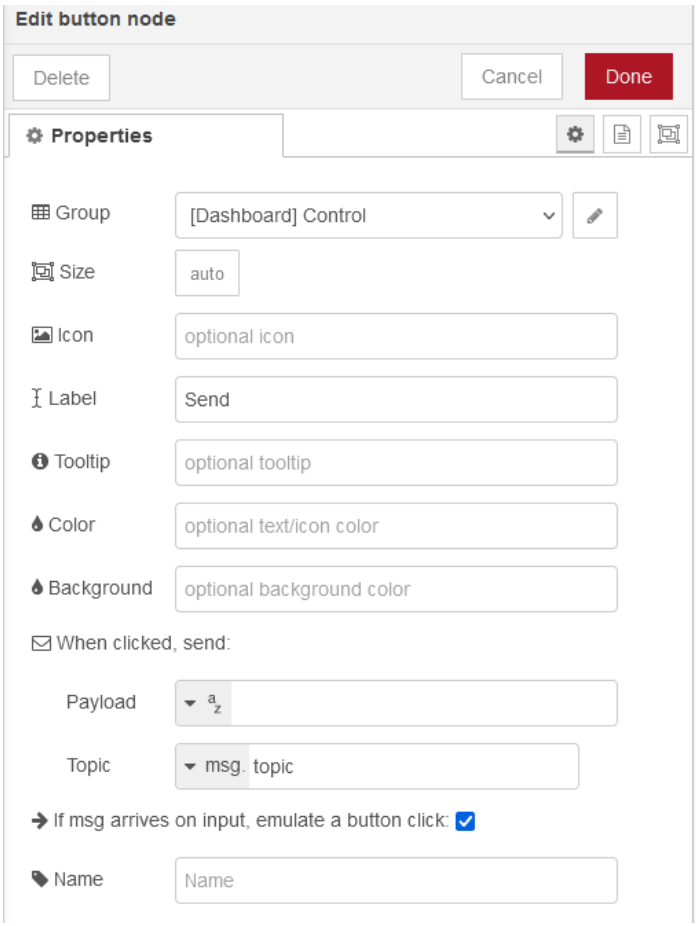


Figure 27: The edit button dialogue

Notice that the output of the button should be statically set. For this reason, we will use a global variable. Global variables are variables that are set globally by some node and can be accessed by every other node. The most common way to set a global variable is through the change node of the function list.

Now insert a change node in the place of the mqtt-out node of the previous example like in Figure 28.

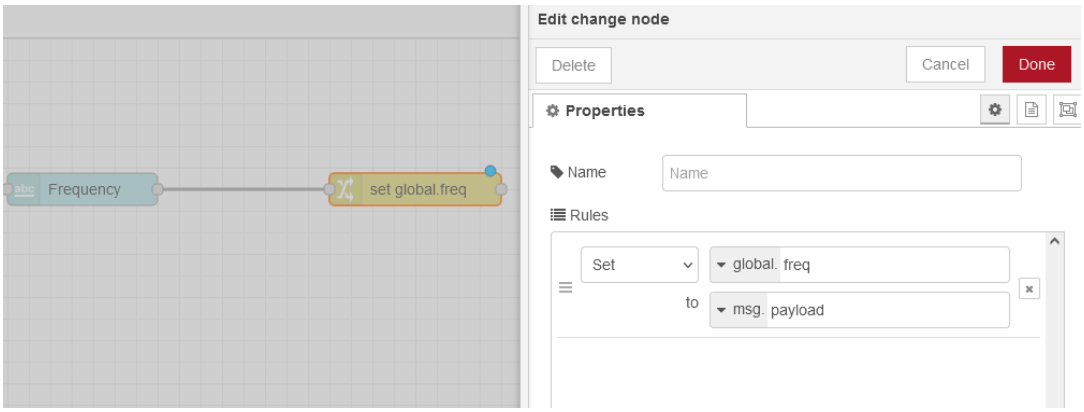
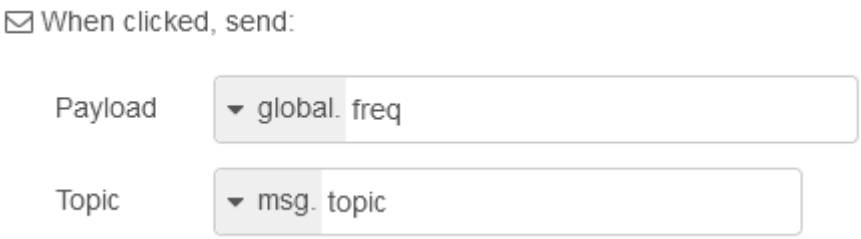


Figure 28: Editing a change node

That way every time the input node sends a message, the value will be saved on the global.freq variable. As such we can set the button paylod to be global.freq, like so:



The completed flow should look like Figure 29.

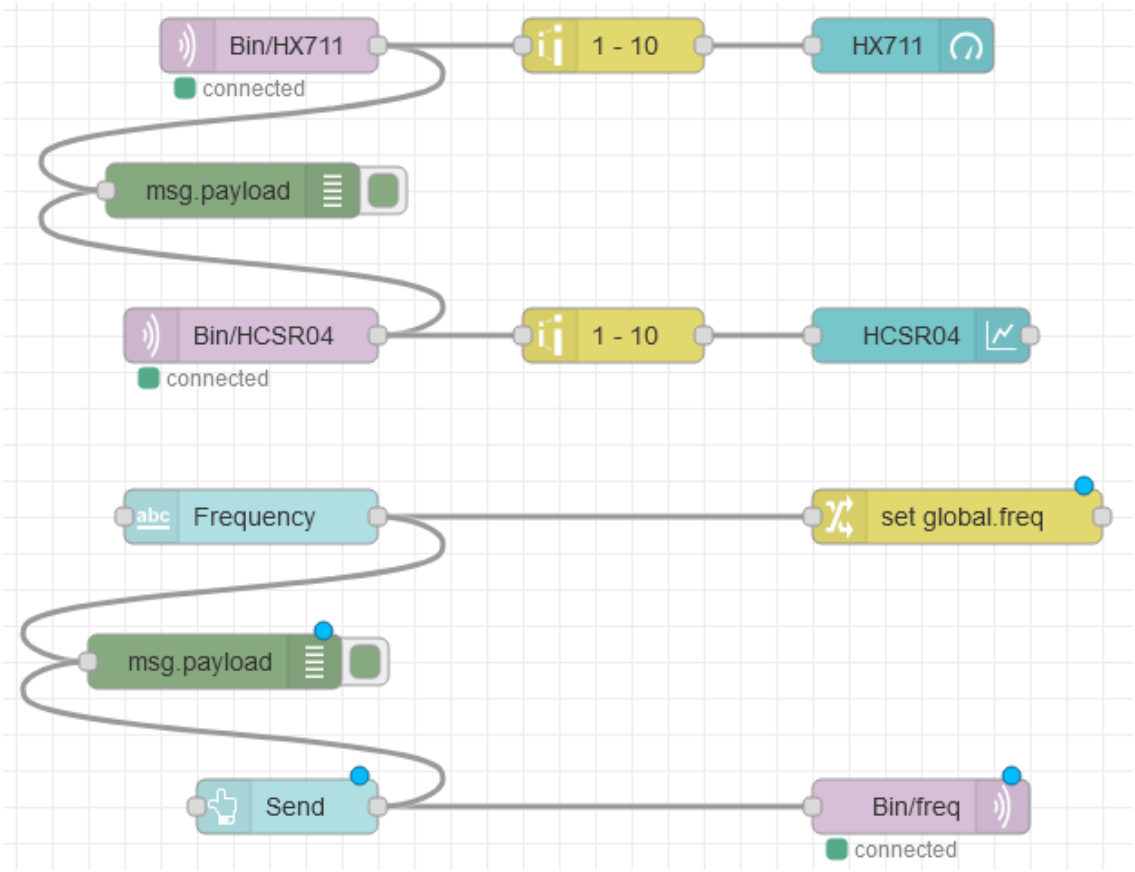


Figure 29: New flow with the send button

So now, the input node sets the global variable freq. and the button, when pressed sends that value to the mqtt server.

Notice that in the previous example, the new frequency sub-flow and the previous dashboard sub flow are not connected. We have a flow that consists of two disjoint parts that is working together. There is no rule that dictates that the nodes should form a connected graph.

The Dashboard now should look like this:

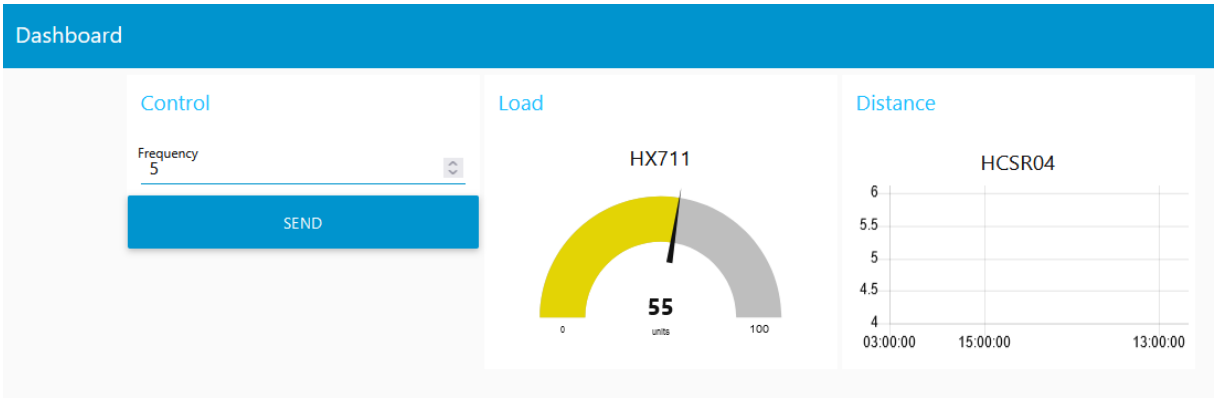


Figure 30: Updated dashboard with send button

In case that the layout is different in your example you can change that in the dashboard view on the right-hand menu (Figure 31).

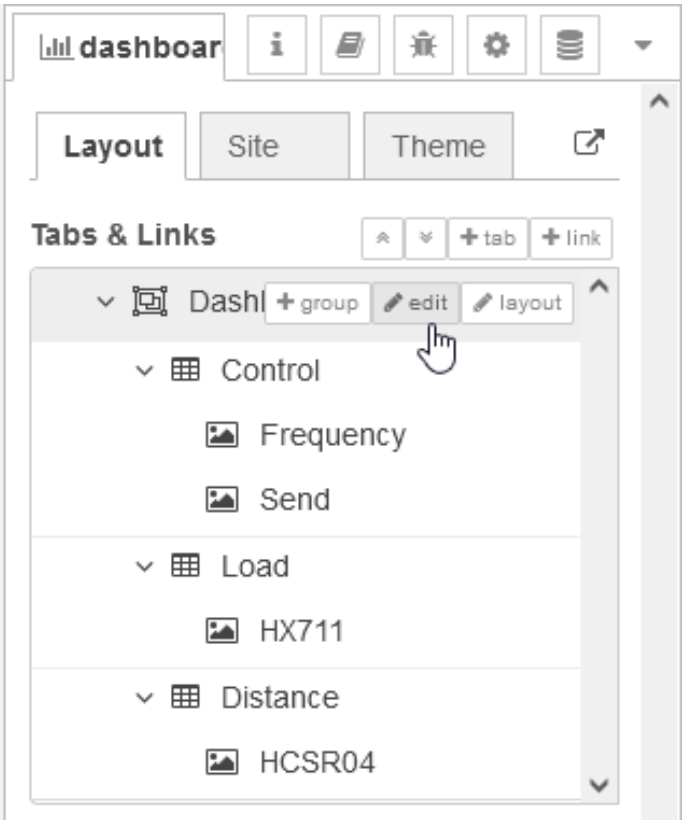


Figure 31: Editing the dashboard layout

Step 2: On and Off

From now on, we will frequently use elements that we have seen in previous steps. For that reason, the descriptions will be faster and not covering every detail, unless some completely new functionality is presented. If you have any problems, feel free to reach one of the instructors and refer to the previous steps.

The next functionality we would like to add is for a way to disable or enable the Bin in total. Firstly, we will implement it so that the user can do it and then we will implement an automatic functionality in case the bin is full or almost full.

The gateway listens for Boolean values in the topic Bin/status and sets its status accordingly. So, we need a way to set and send Boolean values to that topic. The easiest way is to use a switch (dashboard) node, as we have seen in Step 1 of Exercise 2, together with an mqtt-out node, like in Figure 32.

Figure 32: Switch node attached to bin/status

Be sure to set the various parameters correctly (i.e., Dashboard Group, the topic to subscribe, the broker, etc.). Also, as the status message Is not ephemeral, set the retain flag of the MQTT to True.

We have built the manual functionality that easy. Now to implement the automatic functionality we should check the measurements that we have received in Exercise 2. As we have discussed to build flows based on decisions, we will use the switch (function) node (see Step 3 of Exercise 1).

So, the new part of this flow should look like Figure 33.

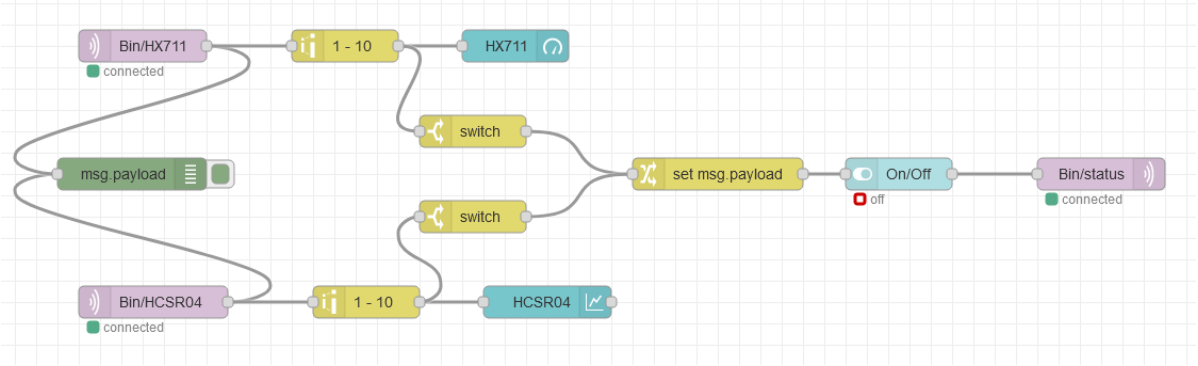


Figure 33: The updated flow with bin/status

With the middle change node setting the payload to false.

In the switch nodes, we should check if the measurements are over (for load) or under (for distance) some threshold and if it is then a message is send to the change node that sets the payload to false and sends it to the switch (dashboard) node.

To check the functionality, as most measurements sent will be inside the threshold, you can use and inject node to send specific measurement to the switch (function) node, like we did in Exercise 1.

As we have seen until now, Node-RED is a graphical language. So, when we are creating our flow, it is good to also look how our flow looks, in terms of readability. In the above example, one could argue that the On/Off switch (dashboard) node, could be missed, in the sense that it looks like it is tightly connected to the change node. To solve this issue, we could use the link in and link out nodes, found under the common list. These nodes are purely for presentation reasons, and they create “invisible” links between them. So, we could alter our flow to Figure 34.

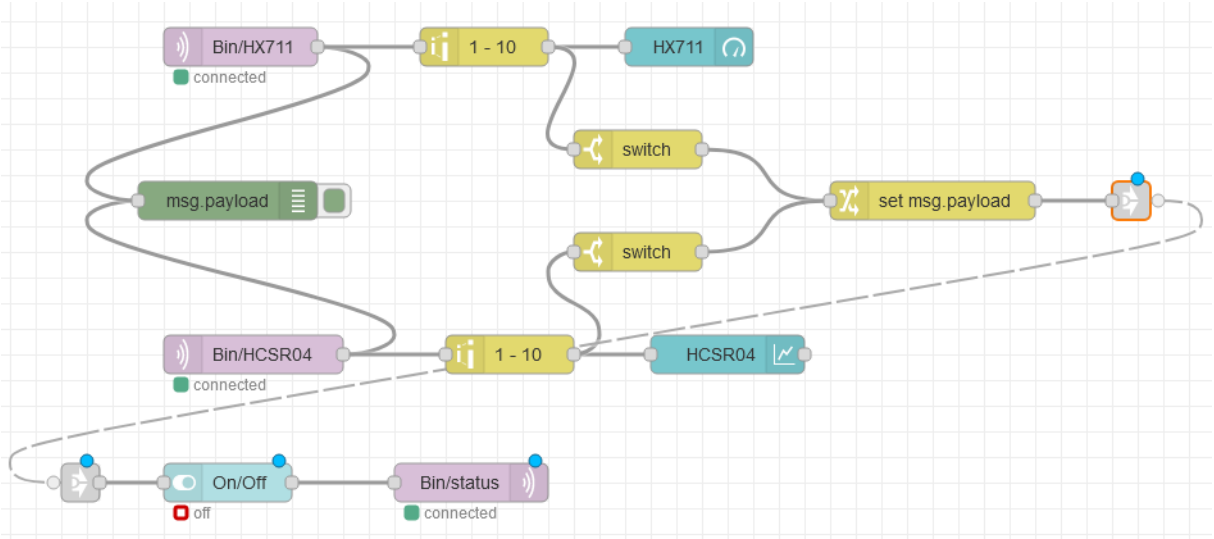


Figure 34: Improving the readability of the flow

The two flows above are exactly the same, but in the second one, the On/Off switch (dashboard) node stands out as an individual item. Be careful however not to overuse link nodes, as they could impede readability in great numbers and bad annotations.

Deploy the flow and check the functionality. The switch must now close automatically when a measurement corresponds to a near full bin. However, from a presentation point of view, it is not enough to just turn off the switch. Such a big change should be presented to the user in a profound way.

Step 3: Presentation

To begin with, we will use the show dialog node, found under the dashboard list. Double clicking it will bring the configuration depicted in Figure 35.

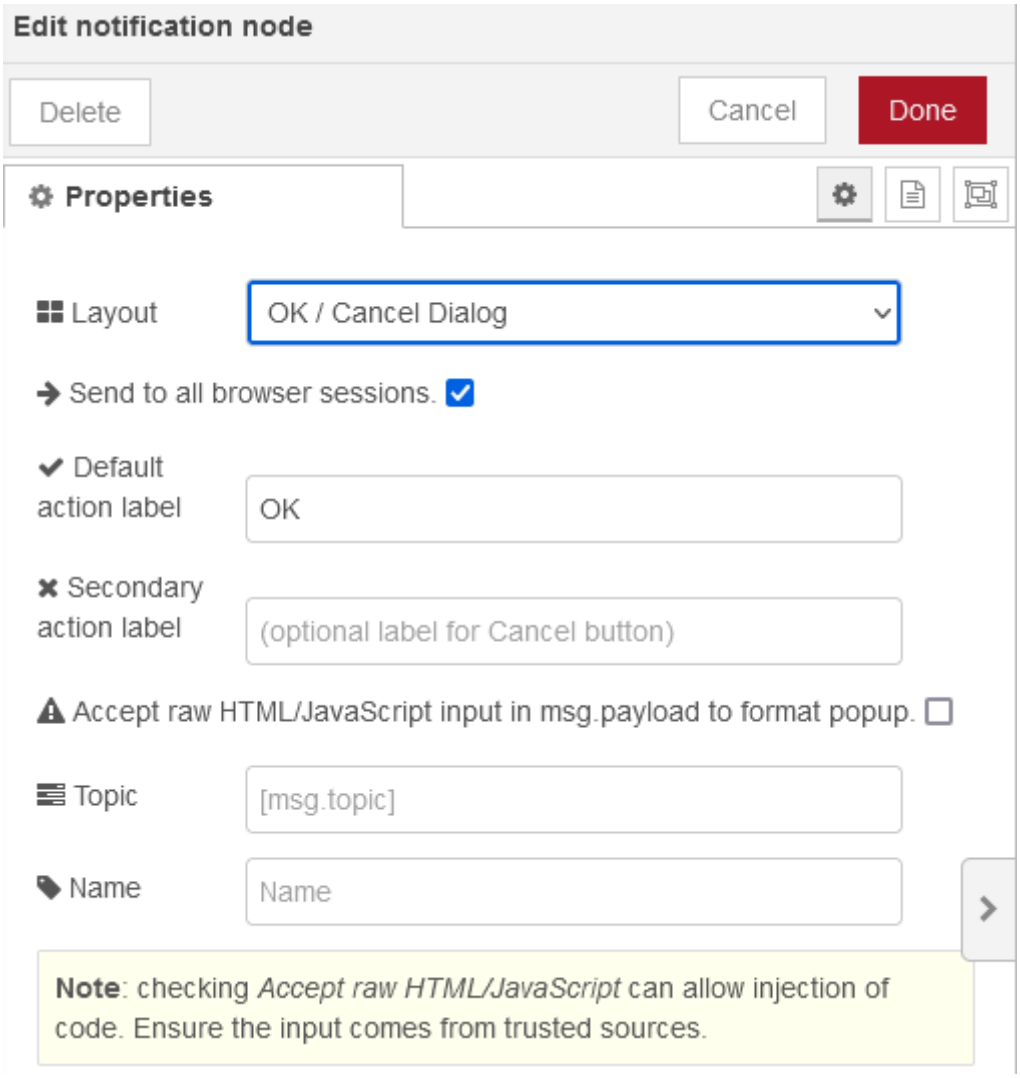
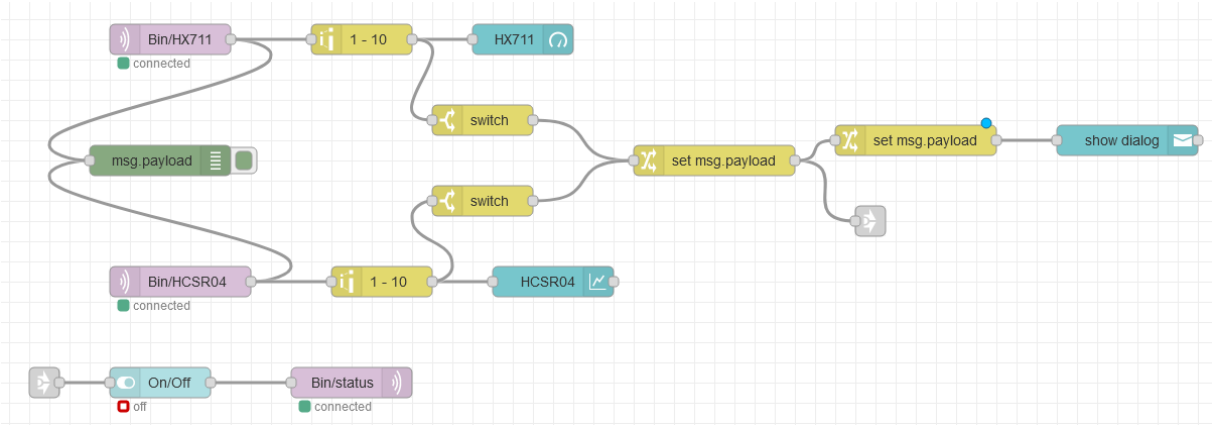


Figure 35: The edit notification node dialogue

Here we can set what kind of Layout we want for the dialog, what actions we want it to have etc. For our case, we want an OK/Cancel dialog, with the cancel label left blank so that there exists only an Ok dialog. That way we know that the user will see it.

We also want to set the message that will be shown. For example, “Bin is Full!”. We will use a change node for that. In total the new flow will look like this

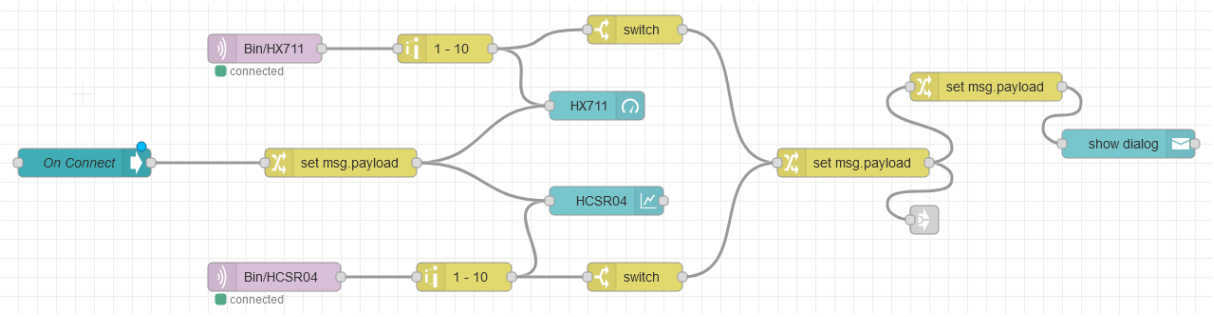


Deploy the flow and try the functionality. A dialog should be shown when the bin is full together with the turn off the Switch.

As a final touch, before concluding this exercise, it will be good to initialize the values properly when a user accesses the dashboard. In Step 1 of the Exercise 1, we saw the ui control node. We will use that to initialize the dashboard components.

For the show dialog node and the button node, no initialization is needed.

For the gauge and the chart node, we will initialize the value to 0 (zero), until we receive a new measurement. That can be easily done by the following flow

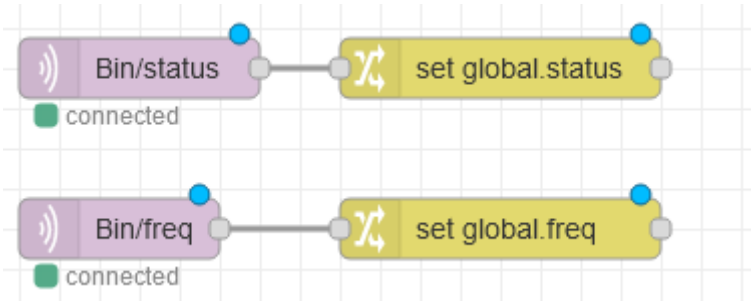


The ui control node fires when a user access the dashboard. Then with the change node, we set the payload to 0 (zero) and we send this “dummy measurement” to the gauge and the chart.

For the text input node and the switch node, it will be a little more difficult. What we want is to show the last value we have in Bin/freq and Bin/status.

Remember from MQTT, to save the last value sent, we must set this message’s retain flag to True! That is why we set it this way earlier.

To do that, we will need two things. First, we need a way to store the last message on the Bin/freq and the Bin/status threads. The easiest way to do that is to use two global variables, which will store the value coming from a subscriber to these topics.

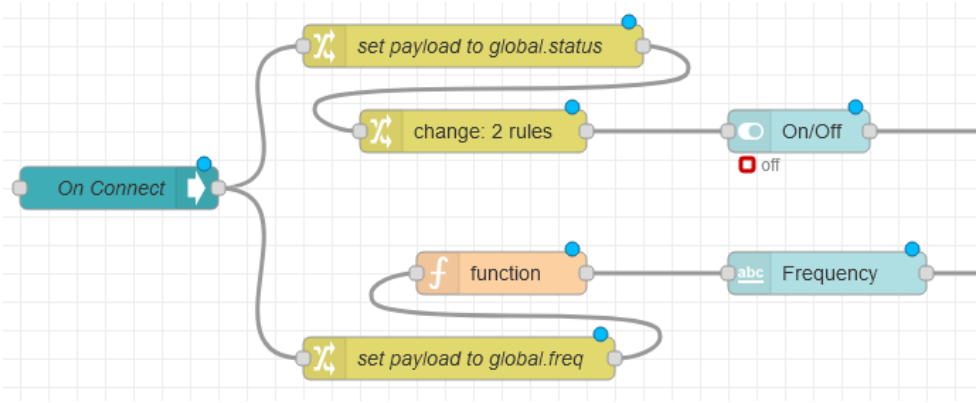


Therefore, we just need to access these variables. The first thought that comes to mind is to do what we did before.

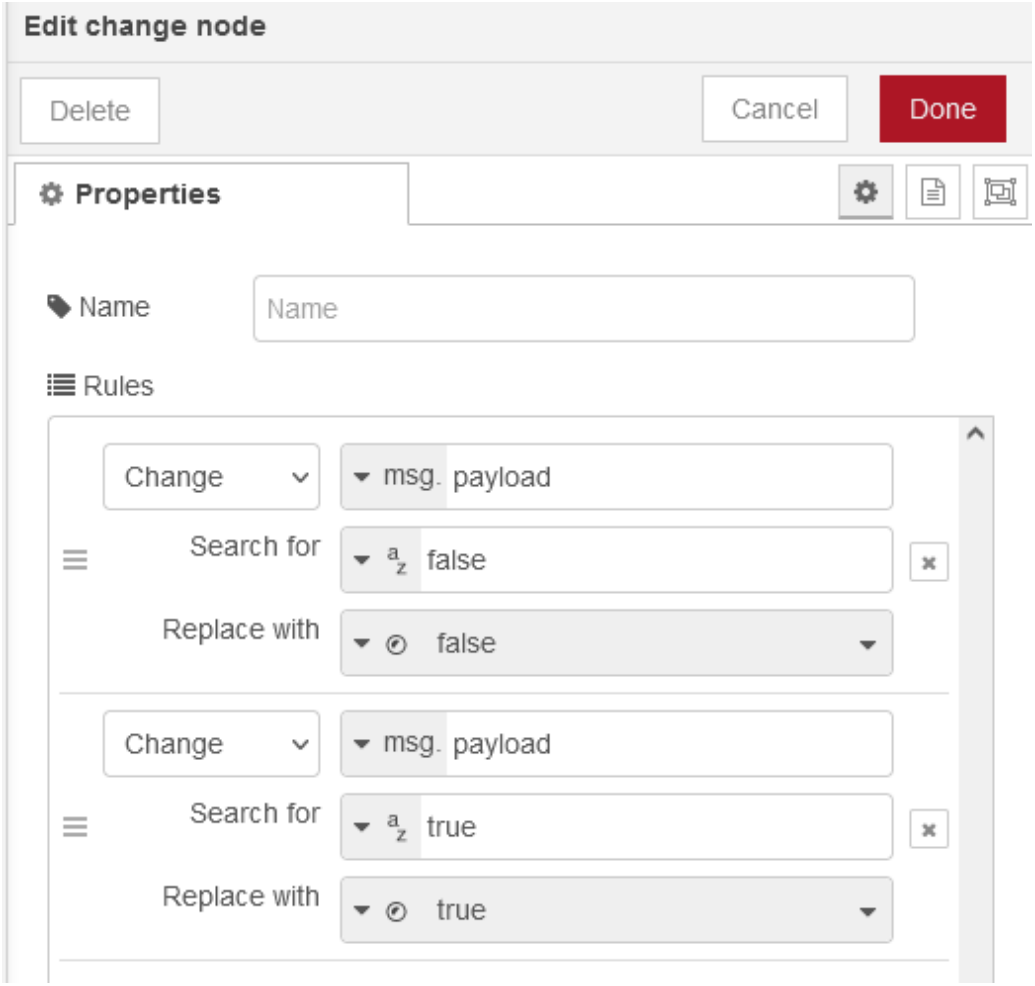
This will not work, however. Try it yourself.

Can you guess why the above flow does not work? Experiment a little before moving forward and see if you understand where the problem is.

The problem is that these variables are stored as strings. While the switch (dashboard) node expects a Boolean input, and the text input node expects a number. So, we will need to cast these two values to the appropriate types before passing them to the two dashboard nodes. We present you two different ways to do that.



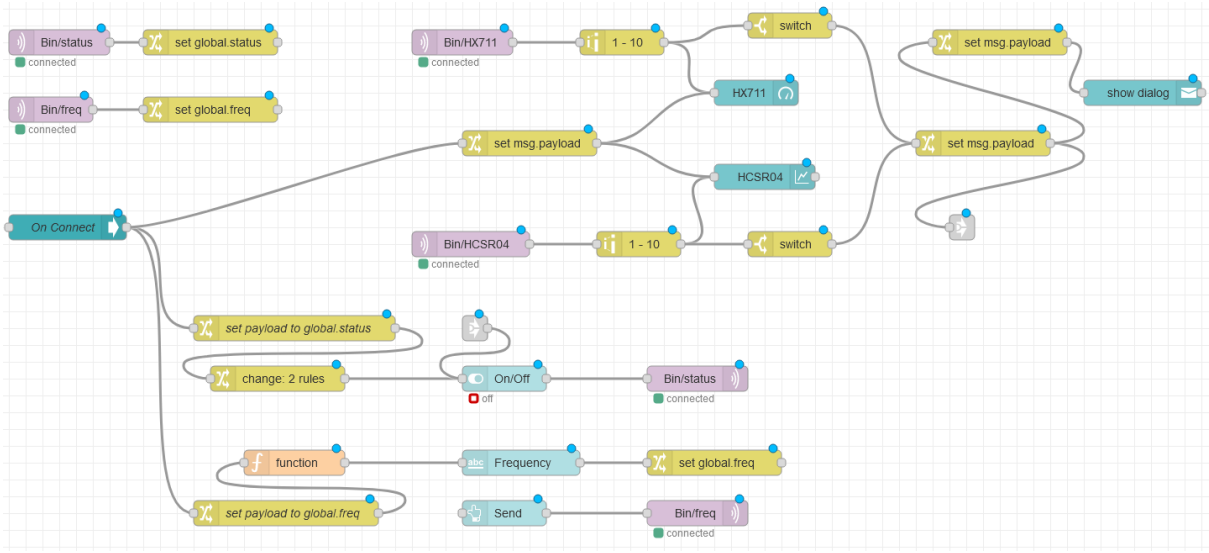
For the status variable we use a change node with two rules as shown below.



For the freq variable, we use a function node with the code below.

```
1  msg.payload = parseInt(msg.payload);
2  return msg;
```

The full flow should look something like this



The JSON code for this flow can be found in the appendix.

As an exercise, try to extend the dashboard above by changing the values of the switch (dashboard) node and the text input node constantly by the retained message on the appropriate topic. This way you would have implemented a dashboard that can be accessed by different users simultaneously.

6. Appendix I: NodeRED project


```
[{"id":"ad92dd32.e2a11","type":"tab","label":"Flow
2","disabled":false,"info":""},{"id":"71c6278e.e2c4a8","type":"mqtt
in","z":"ad92dd32.e2a11","name":"","topic":"Bin/HCSR04","qos":"2","dat
atype":"auto","broker":"4557fb37.e3225c","nl":false,"rap":true,"rh":0,
"x":570,"y":300,"wires":[["f42d6718.64e6e"]]},{"id":"1fd5aa65.7e5446",
"type":"mqtt
in","z":"ad92dd32.e2a11","name":"","topic":"Bin/HX711","qos":"2","data
type":"auto","broker":"4557fb37.e3225c","nl":false,"rap":true,"rh":0,"
x":560,"y":60,"wires":[["d9d82a16.b9285"]]},{"id":"2ed7b0c1.9a357","ty
pe":"ui_gauge","z":"ad92dd32.e2a11","name":"","group":"3f163b2a.61e7a4
","order":1,"width":0,"height":0,"gtype":"gage","title":"HX711","label
":"units","format":"{{value}}","min":"0","max":"100","colors":["#00b50
0","#e6e600","#ca3838"],"seg1":"","seg2":"","x":910,"y":120,"wires":[]
},{"id":"f0ee932b.7fc088","type":"ui_chart","z":"ad92dd32.e2a11","name
":"","group":"a9d9bf65.404188","order":1,"width":0,"height":0,"label":
"HCSR04","chartType":"line","legend":"false","xformat":"HH:mm:ss","int
erpolate":"linear","nodata":"","dot":false,"ymin":"","ymax":"","remove
Older":1,"removeOlderPoints":"","removeOlderUnit":"3600","cutout":0,"u
seOneColor":false,"useUTC":false,"colors":["#1f77b4","#aec7e8","#ff7f0
e","#2ca02c","#98df8a","#d62728","#ff9896","#9467bd","#c5b0d5"],"outpu
ts":1,"useDifferentColor":false,"x":920,"y":220,"wires":[[]]},{"id":"d
9d82a16.b9285","type":"range","z":"ad92dd32.e2a11","minin":"1","maxin"
:"10","minout":"1","maxout":"10","action":"scale","round":true,"proper
ty":"payload","name":"","x":750,"y":60,"wires":[["2ed7b0c1.9a357","9b2
82a0e.46fd28"]]},{"id":"f42d6718.64e6e","type":"range","z":"ad92dd32.e
2a11","minin":"1","maxin":"10","minout":"1","maxout":"10","action":"sc
ale","round":false,"property":"payload","name":"","x":770,"y":300,"wir
es":[["f0ee932b.7fc088","875d85af.8470e"]]},{"id":"c3440c5a.72d68","ty
pe":"ui_button","z":"ad92dd32.e2a11","name":"","group":"c5c4a191.88f97
8","order":2,"width":0,"height":0,"passthru":true,"label":"Send","tool
tip":"","color":"","bgcolor":"","icon":"","payload":"freq","payloadTyp
e":"global","topic":"topic","topicType":"msg","x":610,"y":620,"wires":
[["1814929c.7ddc2d"]]},{"id":"cfd5fe1d.e8c5c","type":"change","z":"ad9
2dd32.e2a11","name":"","rules":[{"t":"set","p":"freq","pt":"global","t
o":"payload","tot":"msg"}],"action":"","property":"","from":"","to":""
,"reg":false,"x":860,"y":560,"wires":[[]]},{"id":"c241c6c4.8775d8","ty
pe":"ui_text_input","z":"ad92dd32.e2a11","name":"","label":"Frequency"
,"tooltip":"Enter the frequency of
measurements","group":"c5c4a191.88f978","order":1,"width":0,"height":0
,"passthru":false,"mode":"number","delay":"300","topic":"topic","topic
Type":"msg","x":630,"y":560,"wires":[["cfd5fe1d.e8c5c"]]},{"id":"18149
29c.7ddc2d","type":"mqtt
out","z":"ad92dd32.e2a11","name":"","topic":"Bin/freq","qos":"","retai
n":"true","respTopic":"","contentType":"","userProps":"","correl":"","
expiry":"","broker":"4557fb37.e3225c","x":840,"y":620,"wires":[]},{"id
":"8b167a0b.5ddc78","type":"ui_switch","z":"ad92dd32.e2a11","name":"",
"label":"On/Off","tooltip":"","group":"c5c4a191.88f978","order":2,"wid
th":0,"height":0,"passthru":true,"decouple":"false","topic":"topic","t
opicType":"msg","style":"","onvalue":"true","onvalueType":"bool","onic
on":"","oncolor":"","offvalue":"false","offvalueType":"bool","officon"
:"","offcolor":"","animate":false,"x":610,"y":460,"wires":[["f2c91f3c.
14a28"]]},{"id":"f2c91f3c.14a28","type":"mqtt
out","z":"ad92dd32.e2a11","name":"","topic":"Bin/status","qos":"","ret
ain":"true","respTopic":"","contentType":"","userProps":"","correl":""
```

```
,"expiry":"","broker":"4557fb37.e3225c","x":840,"y":460,"wires":[]},{"
id":"9b282a0e.46fd28","type":"switch","z":"ad92dd32.e2a11","name":"","
property":"payload","propertyType":"msg","rules":[{"t":"gt","v":"8","v
t":"str"}],"checkall":"true","repair":false,"outputs":1,"x":950,"y":40
,"wires":[["6c8b20ca.f14448"]]},{"id":"875d85af.8470e","type":"switch"
,"z":"ad92dd32.e2a11","name":"","property":"payload","propertyType":"m
sg","rules":[{"t":"gt","v":"8","vt":"str"}],"checkall":"true","repair"
:false,"outputs":1,"x":950,"y":300,"wires":[["6c8b20ca.f14448"]]},{"id
":"6c8b20ca.f14448","type":"change","z":"ad92dd32.e2a11","name":"","ru
les":[{"t":"set","p":"payload","pt":"msg","to":"false","tot":"bool"}],
"action":"","property":"","from":"","to":"","reg":false,"x":1180,"y":1
80,"wires":[["656b981.8ddcfe8","b6dbc76.9456838"]]},{"id":"656b981.8dd
cfe8","type":"link            out","z":"ad92dd32.e2a11","name":"Switch
off","links":["2957e1c6.c762b6"],"x":1155,"y":280,"wires":[]},{"id":"7
5289ce6.90a2cc","type":"ui_toast","z":"ad92dd32.e2a11","position":"dia
log","displayTime":"3","highlight":"","sendall":true,"outputs":1,"ok":
"OK","cancel":"","raw":false,"topic":"","name":"","x":1370,"y":120,"wi
res":[[]]},{"id":"b6dbc76.9456838","type":"change","z":"ad92dd32.e2a11
","name":"","rules":[{"t":"set","p":"payload","pt":"msg","to":"Bin  is
Full!","tot":"str"}],"action":"","property":"","from":"","to":"","reg"
:false,"x":1200,"y":60,"wires":[["75289ce6.90a2cc"]]},{"id":"ae1ba22c.
1717c8","type":"change","z":"ad92dd32.e2a11","name":"","rules":[{"t":"
set","p":"payload","pt":"msg","to":"0","tot":"num"}],"action":"","prop
erty":"","from":"","to":"","reg":false,"x":640,"y":180,"wires":[["2ed7
b0c1.9a357","f0ee932b.7fc088"]]},{"id":"fedb6f9f.09fa8","type":"mqtt
in","z":"ad92dd32.e2a11","name":"","topic":"Bin/status","qos":"2","dat
atype":"auto","broker":"4557fb37.e3225c","nl":false,"rap":true,"rh":0,
"x":80,"y":60,"wires":[["3ddce1b0.1b3146"]]},{"id":"a4e9b231.6f5eb8","
type":"mqtt
in","z":"ad92dd32.e2a11","name":"","topic":"Bin/freq","qos":"2","datat
ype":"auto","broker":"4557fb37.e3225c","nl":false,"rap":true,"rh":0,"x
":70,"y":140,"wires":[["3589dd10.948f42"]]},{"id":"3589dd10.948f42","t
ype":"change","z":"ad92dd32.e2a11","name":"","rules":[{"t":"set","p":"
freq","pt":"global","to":"payload","tot":"msg"}],"action":"","property
":"","from":"","to":"","reg":false,"x":260,"y":140,"wires":[[]]},{"id"
:"3ddce1b0.1b3146","type":"change","z":"ad92dd32.e2a11","name":"","rul
es":[{"t":"set","p":"status","pt":"global","to":"payload","tot":"msg"}
],"action":"","property":"","from":"","to":"","reg":false,"x":260,"y":
60,"wires":[[]]},{"id":"7257ee6b.04e8c8","type":"change","z":"ad92dd32
.e2a11","name":"set                payload                to
global.status","rules":[{"t":"set","p":"payload","pt":"msg","to":"stat
us","tot":"global"}],"action":"","property":"","from":"","to":"","reg"
:false,"x":360,"y":400,"wires":[["3bcdcf5e.f85728"]]},{"id":"3bcdcf5e.
f85728","type":"change","z":"ad92dd32.e2a11","name":"","rules":[{"t":"
change","p":"payload","pt":"msg","from":"false","fromt":"str","to":"fa
lse","tot":"bool"},{"t":"change","p":"payload","pt":"msg","from":"true
","fromt":"str","to":"true","tot":"bool"}],"action":"","property":"","
from":"","to":"","reg":false,"x":340,"y":460,"wires":[["8b167a0b.5ddc7
8"]]},{"id":"f8d05108.381f68","type":"function","z":"ad92dd32.e2a11","
name":"","func":"msg.payload     =     parseInt(msg.payload);\nreturn
msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","libs":[],"x
":360,"y":560,"wires":[["c241c6c4.8775d8"]]},{"id":"f1107447.60fc98","
type":"change","z":"ad92dd32.e2a11","name":"set       payload       to
global.freq","rules":[{"t":"set","p":"payload","pt":"msg","to":"freq",
"tot":"global"}],"action":"","property":"","from":"","to":"","reg":fal
se,"x":350,"y":620,"wires":[["f8d05108.381f68"]]},{"id":"6a962a90.8637
04","type":"ui_ui_control","z":"ad92dd32.e2a11","name":"On
```

```
Connect","events":"connect","x":90,"y":280,"wires":[["7257ee6b.04e8c8"
,"f1107447.60fc98","ae1ba22c.1717c8"]]},{"id":"2957e1c6.c762b6","type"
:"link                          in","z":"ad92dd32.e2a11","name":"Bin
Full","links":["656b981.8ddcfe8"],"x":575,"y":400,"wires":[["8b167a0b.
5ddc78"]]},{"id":"4557fb37.e3225c","type":"mqtt-
broker","name":"","broker":"169.254.62.220","port":"1883","clientid":"
","usetls":false,"compatmode":false,"protocolVersion":"4","keepalive":
"60","cleansession":true,"birthTopic":"","birthQos":"0","birthPayload"
:"","birthMsg":{},"closeTopic":"","closeQos":"0","closePayload":"","cl
oseMsg":{},"willTopic":"","willQos":"0","willPayload":"","willMsg":{},
"sessionExpiry":""},{"id":"3f163b2a.61e7a4","type":"ui_group","name":"
Load","tab":"66d8cfd9.70d3f8","order":2,"disp":true,"width":"6","colla
pse":false},{"id":"a9d9bf65.404188","type":"ui_group","name":"Distance
","tab":"66d8cfd9.70d3f8","order":3,"disp":true,"width":"6","collapse"
:false},{"id":"c5c4a191.88f978","type":"ui_group","name":"Control","ta
b":"66d8cfd9.70d3f8","order":1,"disp":true,"width":"6","collapse":fals
e},{"id":"66d8cfd9.70d3f8","type":"ui_tab","name":"Dashboard","icon":"
dashboard","order":1,"disabled":false,"hidden":false}]
```

7. Appendix II: Circuit sketch and source code

The circuit of this project is provided in Figure 36.

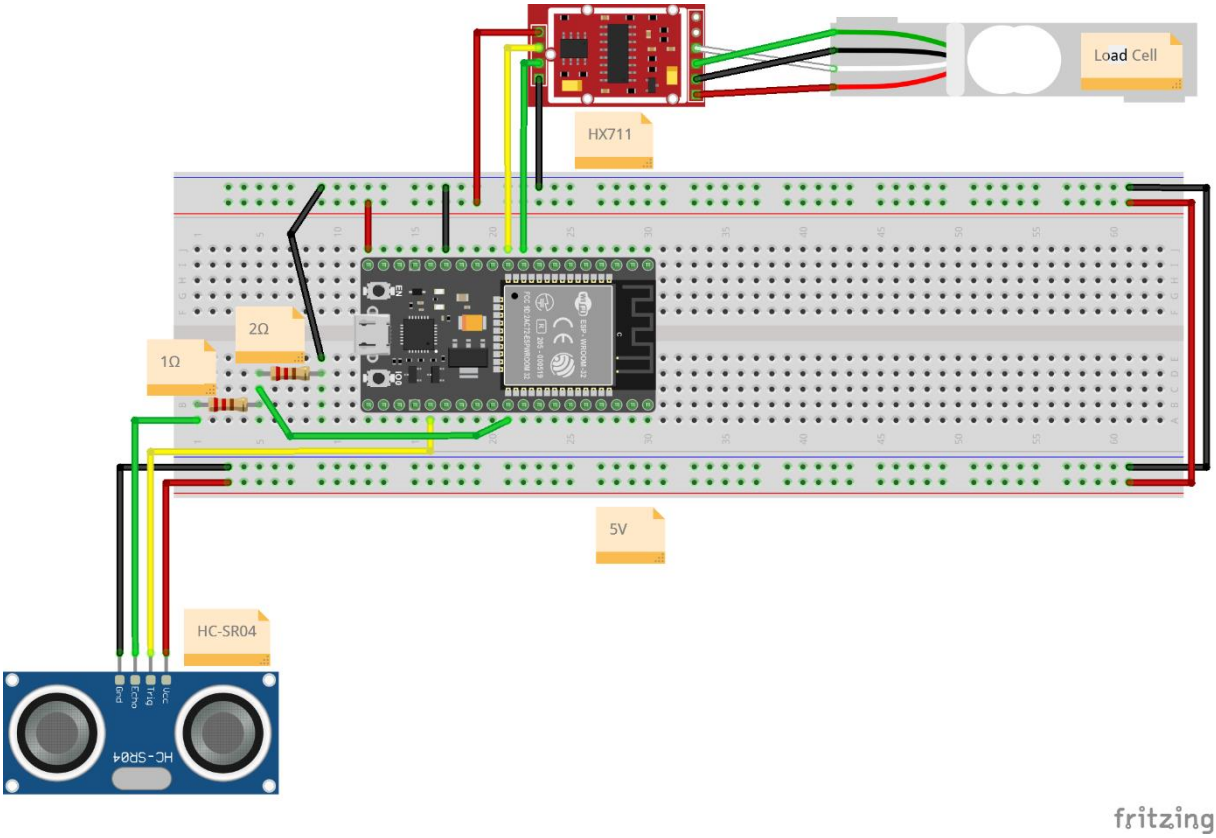


Figure 36: Circuit sketch

The following source code is loaded into the ESP32 device:

```
#include <Arduino.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <SPI.h>
#include <HX711.h>

// HX711 Load Cell
const int LOADCELL_DOUT_PIN = 25;
const int LOADCELL_SCK_PIN = 26;

// HC-SR04 Ultra Sonic Sensor
const int trigPin = 2;
const int echoPin = 5;

// Create Sensor Instances
```

```
// HX711 Load Cell
HX711 scale;

// Defines Variables
// HC-SR04 Ultra Sonic Sensor
long duration;
int distance;

// WiFi
const char* ssid = "";
const char* password = "";
WiFiClient espClient;

// MQTT
const char* mqtt_server = "";
PubSubClient client(espClient);

// Variables for measurements
int ticks = 0;
int freq = 1;
int rst = 0;
char str[256];

// Function that connects to the WiFi
void initWiFi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }
  Serial.println("\nWiFi Connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

// Callback function in case of message arrived
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");

  // Create String from byte array
  char message[length + 1];
  memcpy(message, payload, length);
  message[length] = '\0';

  Serial.println(message);
```

```
  // Check the topic of the message arrived
  if (strcmp (topic, "Bin1/freq") == 0) {
    // Set the frequency of the measurements
    freq = atoi((char *) message);
  }
}

// Function that connect to the MQTT Server
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("ESP8266Client")) {
      Serial.println("connected");
      // When connected to the MQTT Server, delcare subscriptions\
      client.subscribe("Bin1/freq");
    } else {
      // If it fails to connect print error message and retry
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup() {
  // HX711 Load Cell
  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);

  // HC-SR04 Ultra Sonic Sensor
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input

  Serial.begin(9600); // Starts the serial communication

  // Connect to WiFi
  initWiFi();

  // Set MQTT client
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {
  // Make sure we are connected to MQTT server and reconnect if needed
  if (!client.connected()) {
    reconnect();
```

```
  }
  // Check for any new messages
  client.loop();

  // Reset position after 2 seconds without the RFID card
  if (ticks >= freq) {
    // If enought ticks have passed then take measurements
    ticks = 0;
    if (scale.wait_ready_timeout(1000)) {
      long reading = scale.read();
      Serial.print("HX711 reading: ");
      Serial.println(reading);
      sprintf(str, "%ld", reading);
      client.publish("Bin1/HX711", str);
    } else {
      Serial.println("HX711 not found.");
    }
    // Clears the trigPin
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    // Sets the trigPin on HIGH state for 10 micro seconds
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    // Reads the echoPin, returns the sound wave travel time in microseconds
    duration = pulseIn(echoPin, HIGH);
    // Calculating the distance
    float distance = duration*0.034/2;
    // Prints the distance on the Serial Monitor
    Serial.print("Distance: ");
    Serial.println(distance);
    sprintf(str, "%f", distance);
    client.publish("Bin1/HC-SR04", str);
  } else {
    ticks++;
  }

  //Wait for 1 sec (as such 1 tick == 1 sec)
  delay(1000);
}
```

8. References

[1] K. Nirde, P. S. Mulay and U. M. Chaskar, “IoT based solid waste management system for smart city,” in *International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2017.

[2] C. Rattanapoka, S. Chanthakit, A. Chimchai and A. Sookkeaw, “An MQTT-based IoT Cloud Platform with Flow Design by Node-RED,” in *Research, Invention, and Innovation Congress (RI2C)*, Bangkok, Thailand, 2019.

[3] S. Ekelof, “The genesis of the Wheatstone bridge,” *Engineering Science and Education Journal,* vol. 10, no. 1, pp. 37-40, 2021.

[4] SparkFun Electronics, “SparkFun Load Cell Amplifier - HX711,” [Online]. Available: https://www.sparkfun.com/products/13879. [Accessed 28 April 2022].

[5] A. A. Jabbaar, “Ultrasonic Sensor HC-SR04 with Arduino Tutorial,” 17 September 2019. [Online]. Available: https://create.arduino.cc/projecthub/abdularbi17/ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6. [Accessed 28 April 2022].

9. Contacts

Project Coordinator:

- Name: Technical University of Sofia
- Address:
    - Technical University of Sofia,
      Kliment Ohridsky Bd 8
      1000, Sofia, Bulgaria
- Phone: +3592623073

Output 2 Leader:

- Name: FOSS Research Centre for Sustainable Energy, University of Cyprus
- Address:
    - University of Cyprus,
      Panepistimiou 1 Avenue
      P.O. Box 20537
      1678, Nicosia, Cyprus
- Email: foss@ucy.ac.cy
- Phone: +357 22 894288