

Projeto Domótica - Internet das Coisas

a39041 - José Borges

Bragança, Junho 2023

Introdução

O projeto Domótica proposto é um trabalho acadêmico desenvolvido no âmbito da disciplina de Internet das Coisas. O objetivo principal do projeto é aplicar os conceitos e habilidades aprendidos ao longo da disciplina, colocando em prática os princípios da comunicação entre dispositivos, coleta de dados de sensores, processamento desses dados e sua visualização, bem como a integração com serviços externos.

Domótica é um termo que se refere à automação num ambiente, por exemplo residencial ou empresarial, que utiliza a tecnologia para controlar e automatizar diversos sistemas e dispositivos. É possível controlar e automatizar uma variedade de sistemas, como iluminação, climatização e cortinas. Esses sistemas podem ser acessados e controlados localmente, ou remotamente, utilizando uma conexão à internet.

A domótica também visa aprimorar a eficiência energética, otimizando o uso de recursos e reduzindo o consumo desnecessário.

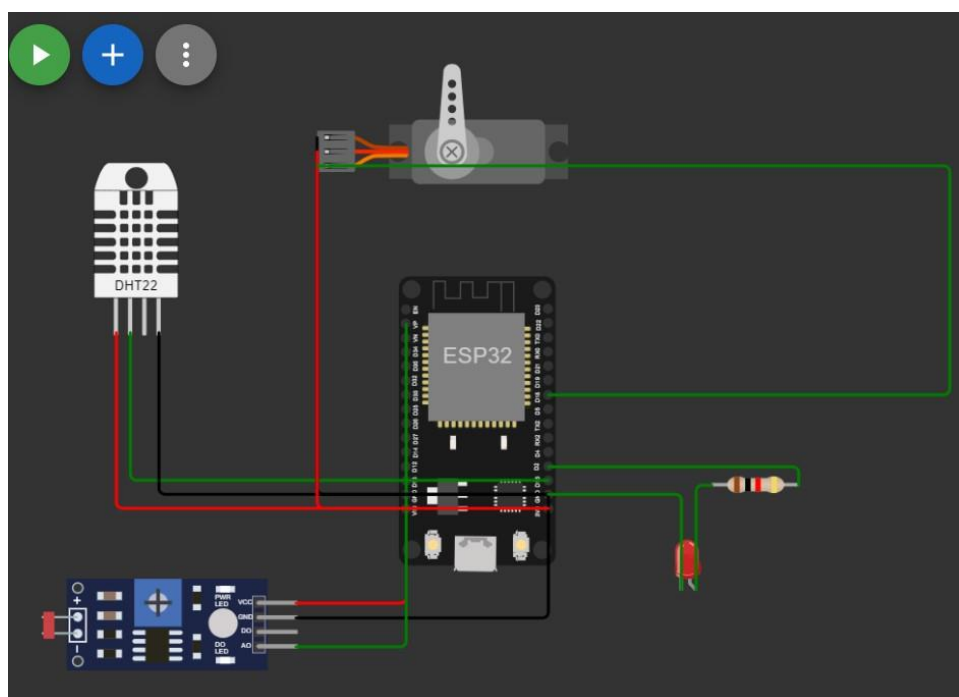
Contextualização

O espaço alvo para a simulação deste trabalho é uma sala de estar, onde serão aplicados os conceitos e tecnologias para criar o sistema de domótica simulado. A sala de estar foi escolhida como ambiente devido à sua relevância como espaço, permitindo uma aplicação prática e significativa das funcionalidades. Serão utilizados sensores para medir a quantidade de luz, a concentração de CO₂, a temperatura e a humidade do ar, bem como atuadores, servo motor para controle do estado da janela e um LED a serem automatizados.

Neste contexto, o ambiente da sala de estar será simulado utilizando a plataforma online Wokwi (<https://wokwi.com>), uma ferramenta que permite a simulação de circuitos eletrônicos e sistemas baseados em microcontroladores de forma virtual, utilizando

componentes e dispositivos virtuais.

Isso significa que os sensores, como o LDR (sensor de luz), o DHT22 (sensor de temperatura e humidade) e os atuadores, servo motor e LED, assim como o controlador central ESP32, serão representados digitalmente na plataforma permitindo visualizar e interagir com os componentes de forma virtual, oferecendo uma representação visual realista do sistema.



Além disso, o projeto considera a informação sobre o nascer e o pôr do sol, que pode ser obtida de fontes externas, como o OpenWeather e o Instituto Português do Mar e da Atmosfera (IPMA).

O sistema de domótica na sala de estar permitirá o controle da janela e de um LED, monitorar a temperatura e ventilação do ambiente, bem como a qualidade do ar com base na concentração de CO₂. O sistema também contará com atuadores simulados para controlar a cortina em diferentes posições (alta, média e baixa), a janela em diferentes aberturas (semi-aberta, aberta e fechada) e o ar-condicionado em diferentes modos (ligado aquecendo, arrefecendo, ventilando e desligado).

Os dados coletados pelos sensores e as informações externas serão exibidos em uma dashboard no node-red, fornecendo informações como a duração do dia e os horários de nascer e pôr do sol (histórico dos últimos 7 dias), a qualidade do ar interna, as temperaturas e humidades internas e externas, as posições da cortina e janela, e o estado do funcionamento do ar condicionado.

Desenvolvimento:

1. Informação do Nascer e pôr do sol do dia atual.

No fluxo do Node-RED, o processo descrito tem como objetivo obter os dados meteorológicos do dia atual, por meio da API do OpenWeather (current weather).



Começa-se com a inicialização do nó "timestamp", que é configurado para fazer um inject a cada 24 horas. Em seguida, é utilizado o nó "http-request" para fazer uma solicitação à API do OpenWeather, com o objetivo de obter os dados com a informação do nascer e pôr do sol.

Após fazer a requisição à API e receber a resposta em JSON, os dados de nascer e pôr do sol são extraídos e manipulados utilizando uma série de operações, utilizando o nó de função.

```

1
2  const diasSemana = ['Domingo', 'Segunda-feira', 'Terça-feira', 'Quarta-feira',
3    'Quinta-feira', 'Sexta-feira', 'Sábado'];
4
5  const dataHoje = new Date(msg.payload.dt * 1000);
6  const sunrise = new Date(msg.payload.sys.sunrise * 1000);
7  const sunset = new Date(msg.payload.sys.sunset * 1000);
8
9  const year = dataHoje.getFullYear();
10 const month = dataHoje.getMonth() + 1;
11 const day = dataHoje.getDate();
12
13 const weekNumber = dataHoje.getDay();
14 const diaDaSemana = diasSemana[weekNumber];
15
16 const data = `${day}-${month}-${year}`;
17
18 const sunriseString = `${sunrise.getHours()}:${sunrise.getMinutes()}`;
19 const sunsetString = `${sunset.getHours()}:${sunset.getMinutes()}`;
20
21 msg.payload = {
22   week: diaDaSemana,
23   dataHoje: data,
24   sunrise: sunriseString,
25   sunset: sunsetString
26 };
27
28 return msg;

```

Primeiramente, são criadas instâncias de objetos "Date" a partir dos valores recebidos da requisição à API do OpenWeather. Em seguida, são extraídas informações como o ano, mês, dia e dia da semana. Os horários de nascer e pôr do sol também são extraídos e formatados para exibição. As horas e minutos são obtidos dos objetos "Date" e convertidos em strings no formato "HH:mm". Os dados processados são armazenados em uma nova estrutura de mensagem "msg.payload", contendo o dia da semana, a data de hoje, o horário de nascer do sol e o horário de pôr do sol.

Após a função, o fluxo continua com o nó "ui-template" do pacote "node-red-dashboard", que é responsável por exibir os dados no dashboard.

```

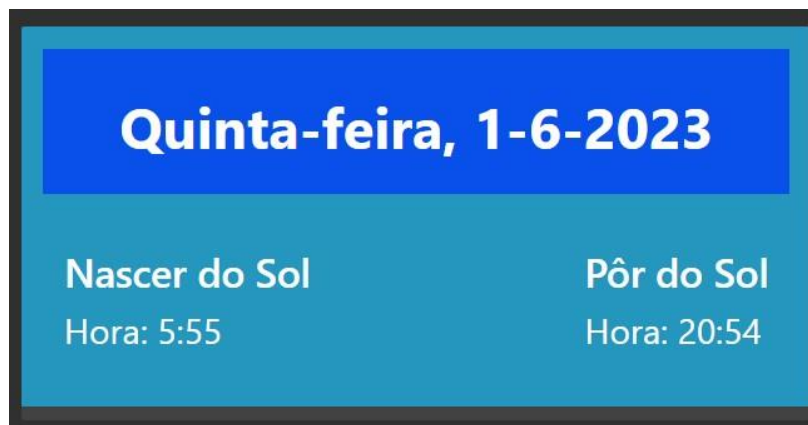
1 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
2 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"></script>
3
4 <div class="container" style="color:white; background-color:#2596be;">
5   <div class="row">
6     <div class="col">
7       <h1 class="mb-1"
8         style="background-color:#0950ea;font-size:40px;text-align:center;font-weight:bold;padding:30px 15px 25px 15px;"
9         {{msg.payload.week+"", "+msg.payload.dataHoje}}></h1>
10
11       <div class="d-flex justify-content-between" style="padding:30px 15px 30px 15px;">
12         <div>
13           <h3>Nascer do Sol</h3>
14           <p style="font-size:25px;">Hora: {{msg.payload.sunrise}}</p>
15         </div>
16         <div>
17           <h3>Pôr do Sol</h3>
18           <p style="font-size:25px;">Hora: {{msg.payload.sunset}}</p>
19         </div>
20       </div>
21     </div>
22   </div>
23 </div>

```

No código do nó "ui-template", são adicionados os links para os estilos e scripts do Bootstrap e é criada uma estrutura em código HTML estilizado com classes e estilos inline para definir cores, tamanho da fonte e formatação do texto.

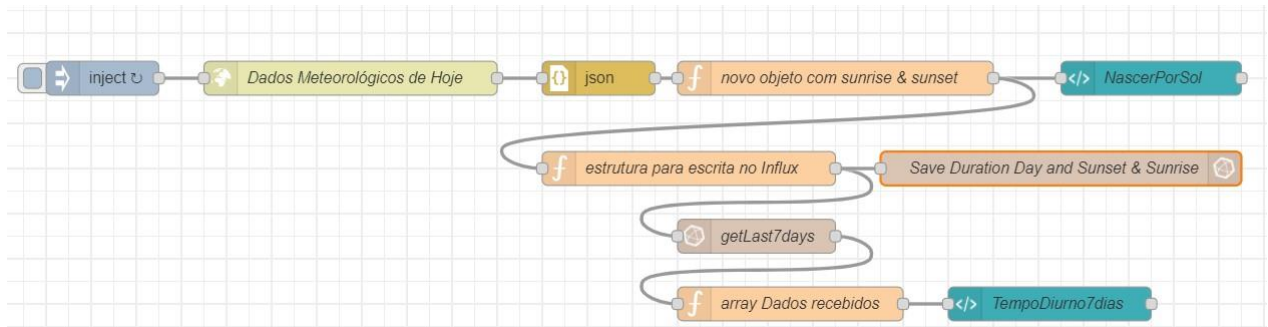
Assim, o nó "ui-template" possibilita a criação de um componente visual personalizado no dashboard do Node-RED, mostrando as informações de nascer e pôr do sol de maneira clara e organizada.

Resultado final no node-red dashboard:



2. Duração do dia (histórico dos últimos 7 dias)

O processo começa com o reaproveitamento do fluxo do Node-RED que lida com informações sobre o nascer e o pôr do sol. A partir do objeto gerado por esse fluxo, é criado um novo nó de função para estruturar a escrita dos dados no InfluxDB.



Os valores de nascer e pôr do sol, juntamente com a data, são extraídos do objeto *msg.payload* da função anterior.

Para calcular a duração do dia, os horários de nascer e pôr do sol são convertidos em números inteiros separando as horas e minutos. Em seguida, são criados objetos do tipo *Date* com base nesses valores.

```
1  //msg.payload
2  var sunriseString = msg.payload.sunrise;
3  var sunsetString = msg.payload.sunset;
4  var data = msg.payload.dataHoje;
5  //converter sunriseString para Date.setHours
6  const sunriseParts = sunriseString.split(':');
7  const sunriseHour = parseInt(sunriseParts[0]);
8  const sunriseMinutes = parseInt(sunriseParts[1]);
9  const sunriseDate = new Date();
10 sunriseDate.setHours(sunriseHour, sunriseMinutes, 0, 0);
11
12 //converter sunsetString para Date.setHours
13 const sunsetPartsSunset = sunsetString.split(':');
14 const sunsetHour = parseInt(sunsetPartsSunset[0]);
15 const sunsetMinutes = parseInt(sunsetPartsSunset[1]);
16 const sunsetDate = new Date();
17 sunsetDate.setHours(sunsetHour, sunsetMinutes, 0, 0);
```

A duração do dia é calculada subtraindo-se o horário de nascer do sol do horário de pôr do sol, resultando em um valor em milissegundos. Esse valor é convertido para horas e minutos separados utilizando funções matemáticas.

```

19 const durationMillis = sunsetDate.getTime() - sunriseDate.getTime();
20 const durationHours = Math.floor(durationMillis / (1000 * 60 * 60));
21 const durationMinutes = Math.floor((durationMillis % (1000 * 60 * 60)) / (1000 * 60));
22
23
24 const duracaoString = durationHours+":"+durationMinutes;
25 console.log(duracaoString);

```

O resultado final é armazenado em uma string no formato "horas:minutos" na variável `duracaoString`. O objeto `msg.payload` é reestruturado para conter as informações relevantes, como os horários de nascer e pôr do sol, a data e a duração do dia.

```

27 msg.payload = [
28   {
29     measurement: "Duração do dia de hoje",
30     fields: {
31       sunrise: sunriseString,
32       sunset: sunsetString,
33       dataHoje: data,
34       duracaoDia: duracaoString
35     },
36     tags: {
37       sensorID: 1,
38       location: "Desk"
39     }
40   }
41 ];
42 return msg;

```

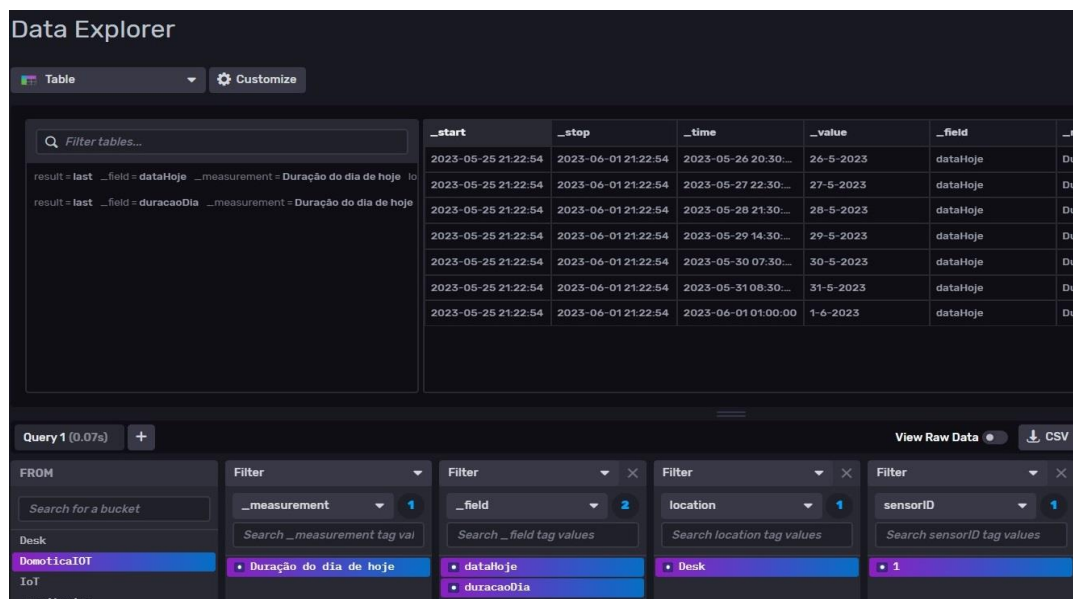
O objeto modificado é então retornado através do fluxo do Node-RED, e posteriormente será enviado ao InfluxDB para armazenamento, utilizando o `influx batch node`. Nessa etapa, é necessário configurar o servidor do InfluxDB e indicar o bucket apropriado para onde os dados serão enviados.

Name	Save Duration Day and Sunset & Sunrise
Server	[v2.0] http://localhost:8086
Organization	IPB
Bucket	DomoticalOT
Time Precision	Milliseconds (ms)

Após armazenar os dados no InfluxDB por meio do nó influxdb o próximo passo no fluxo do Node-RED é obter os dados dos últimos 7 dias dos objetos armazenados na base de dados, e, através do nó influxdb in definimos a seguinte query:

```
1 from(bucket: "DomoticaIoT")
2   |> range(start: -7d)
3   |> filter(fn: (r) => r["_measurement"] == "Duração do dia de hoje")
4   |> filter(fn: (r) => r["_field"] == "duracaoDia" or r["_field"] == "dataHoje" )
5   |> filter(fn: (r) => r["location"] == "Desk")
6   |> filter(fn: (r) => r["sensorID"] == "1")
7   |> aggregateWindow(every: 24h, fn: last, createEmpty: false)
8   |> yield(name: "last")
9
```

A query no InfluxDB realiza várias etapas para filtrar e processar os dados desejados. Primeiro, é especificado o bucket "DomoticalOT" como a fonte de dados e um intervalo de tempo de 7 dias é definido. São aplicados filtros para selecionar medições dos campos "duracaoDia" e "dataHoje" do bucket:



The screenshot shows the InfluxDB Data Explorer interface. At the top, there's a 'Table' view selector and a 'Customize' button. Below that is a search bar for filtering tables. The main area displays a table with columns: _start, _stop, _time, _value, _field, and _measurement. The table contains several rows of data. Below the table, there's a 'Query1 (0.07s)' section with a '+' button. To the right of the query section are buttons for 'View Raw Data' and 'CSV'. Below the query section is a 'FROM' dropdown showing 'Desk', 'DomoticaIoT', and 'IoT'. To the right of the 'FROM' dropdown are four 'Filter' sections. Each filter section has a dropdown menu and a search bar. The filters are: 1. _measurement: Duração do dia de hoje. 2. _field: dataHoje, duracaoDia. 3. location: Desk. 4. sensorID: 1.

_start	_stop	_time	_value	_field	_measurement
2023-05-25 21:22:54	2023-06-01 21:22:54	2023-05-26 20:30:...	26-5-2023	dataHoje	Duração do dia de hoje
2023-05-25 21:22:54	2023-06-01 21:22:54	2023-05-27 22:30:...	27-5-2023	dataHoje	Duração do dia de hoje
2023-05-25 21:22:54	2023-06-01 21:22:54	2023-05-28 21:30:...	28-5-2023	dataHoje	Duração do dia de hoje
2023-05-25 21:22:54	2023-06-01 21:22:54	2023-05-29 14:30:...	29-5-2023	dataHoje	Duração do dia de hoje
2023-05-25 21:22:54	2023-06-01 21:22:54	2023-05-30 07:30:...	30-5-2023	dataHoje	Duração do dia de hoje
2023-05-25 21:22:54	2023-06-01 21:22:54	2023-05-31 08:30:...	31-5-2023	dataHoje	Duração do dia de hoje
2023-05-25 21:22:54	2023-06-01 21:22:54	2023-06-01 01:00:00	1-6-2023	dataHoje	Duração do dia de hoje

O resultado é um array com as entradas filtradas. No próximo nó de função, esse array é processado e decomposto.

```

1  var arrayDatas = [];
2  var arrayDuracaoDia = [];
3  var newJson = {};
4  var finalArray = [];
5
6  var filteredArray = msg.payload.map((obj) => {
7      const { _field, _value } = obj;
8
9      if(_field === "dataHoje"){
10         arrayDatas.push(_value);
11     }else if(_field === "duracaoDia"){
12         arrayDuracaoDia.push(_value);
13     }
14 });
15
16 for (let i = arrayDatas.length-1; i>=0; i--) {
17     newJson = {
18         data : arrayDatas[i],
19         duracao: arrayDuracaoDia[i],
20     }
21     finalArray.push(newJson);
22 }
23
24 msg.payload = finalArray;
25
26 return msg;

```

Os valores de data e duração do dia de cada entrada são armazenados em arrays separados. Um novo array é criado para combinar os valores de data e duração do dia em objetos. Cada objeto é adicionado a um array final. Ao final do processo, o array final contém os objetos com informações de data e duração do dia dos últimos 7 dias, em ordem decrescente. Esse array é atribuído à propriedade `msg.payload`.

De seguida os dados são encaminhados para o nó ui-template usado para criar uma interface de usuário no Node-RED dashboard.

```

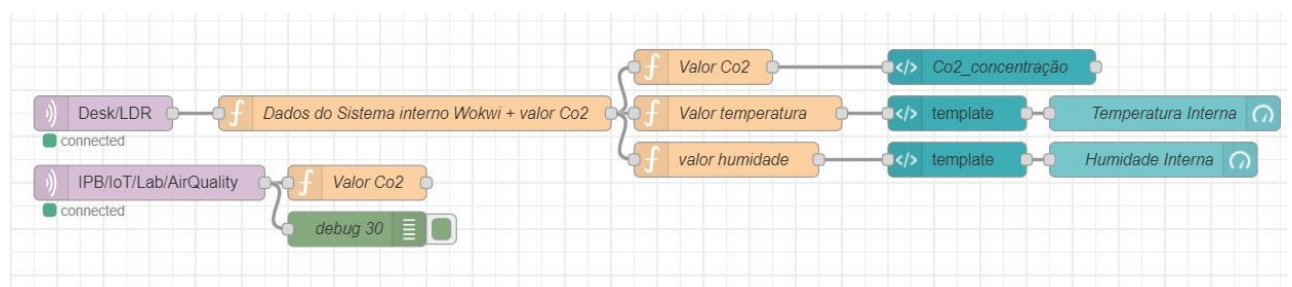
14 <div class="container" style="padding-vertical:9px;">
15     <div class="row">
16         <div class="col">
17             <h3 style="text-align:center;">
18                 Tempo Diurno dos últimos 7 dias (em horas)</h3>
19             <div class="d-flex justify-content-between">
20                 <table class="table table-dark" style="color:white; font-size:20px; ">
21                     <thead>
22                         <tr style="text-align:center" id="header-row">
23                             <th ng-repeat="(data, duracao) in msg.payload[0]">{{data}}</th>
24                         </tr>
25                     </thead>
26                     <tbody>
27                         <tr style="text-align:center" ng-repeat="item in msg.payload" ng-class="{ 'first-row': $index === 0}">
28                             <td ng-repeat="(data, duracao) in item">{{duracao}}</td>
29                         </tr>
30                     </tbody>
31                 </table>
32             </div>
33         </div>
34     </div>
35 </div>

```

O código HTML cria uma tabela que exibe a informação da duração do tempo diurno dos últimos 7 dias. Os dados são exibidos em linhas e colunas, onde cada coluna representa um dia e cada linha representa um atributo do tempo diurno (data e duração). Os valores correspondentes são preenchidos dinamicamente a partir dos dados fornecidos em msg.payload usando de AngularJS. O resultado final do dashboard é mostrado na seguinte imagem:

Tempo Diurno dos últimos 7 dias (em horas)	
data	duracao
1-6-2023	14:59
31-5-2023	14:59
30-5-2023	14:57
29-5-2023	14:56
28-5-2023	14:55
27-5-2023	14:53
26-5-2023	14:52

3. Temperatura e Humidades internas, e concentração de Co2 (qualidade do ar interna)



Neste fluxo, os dados de temperatura e humidade interna são obtidos a partir do sistema simulado no Wokwi utilizando o sensor

DHT22, através do tópico Desk/LDR.

No entanto, devido à impossibilidade de medir diretamente a concentração de CO2 nesse sistema, a medida de co2 é obtida através do tópico "IPB/IoT/Lab/AirQuality" apenas para fins de simulação.

Este valor de Co2 é armazenado em uma variável global, através do nó de função, permitindo que o valor seja acessado e utilizado em outros fluxos.

```
1  const co2 = msg.payload.co2_eqv;  
2  
3  global.set("co2", co2);
```

Depois de subscrever o tópico Desk/LDR e receber o objeto JSON dos dados dos sensores, estes valores são colocados em variáveis globais para acesso mais tarde e agrupados com o valor da variável global de co2, no seguinte nó de função:

```
1  global.set("temp_int", msg.payload.tempei  
2  global.set("hum_int", msg.payload.humidi  
3  global.set("sensor_lux", msg.payload.ldr  
4  
5  if (global.get("co2")){  
6    msg.payload.co2 = global.get("co2");  
7    return msg;  
8  }  
9
```

Desk/LDR : msg.payload : Object

▼ object

ldr_lux: 497.0431
temperature: 21.6
humidity: 77
co2: 2116.67

Após juntar o valor de CO2 ao objeto JSON, o fluxo é dividido em três partes para exibir cada informação separadamente. Para mostrar o valor de CO2, exibido em um container informativo no dashboard, é utilizado o nó "ui-template" com o seguinte código em HTML:

```

1
2 <div class="container"
3   ng-style="{ 'background-color': (msg.payload < 800) ? 'green' : (msg.payload >= 800 && msg.payload <= 1200) ? 'yellow' : (msg
4   <div class="row">
5     <div class="col">
6       <div>
7         <h3 ng-style="{ 'color': (msg.payload >= 800 && msg.payload <= 1200) ? 'black' : 'white' }"
8         style="text-align:center; padding:15px;">Concentração de CO2</h3>
9       </div>
10      <p ng-style="{ 'color': (msg.payload >= 800 && msg.payload <= 1200) ? 'black' : 'white' }"
11      style="text-align:center;font-weight:bold;padding-top:25px; font-size:30px;">
12        {{(msg.payload < 800) ? 'Frac' :
13          (msg.payload >= 800 && msg.payload <= 1200) ? 'Regular' :
14          (msg.payload > 1200 && msg.payload < 1800) ? 'Ruim' : 'Muito Ruim'}}</p>
15      <p ng-style="{ 'color': (msg.payload >= 800 && msg.payload <= 1200) ? 'black' : 'white' }"
16      style="text-align:center;font-weight:bold;padding-top:25px; font-size:30px;">{{msg.payload + ' ppm'}}</p>
17    </div>
18  </div>
19 </div>
20 </div>
21

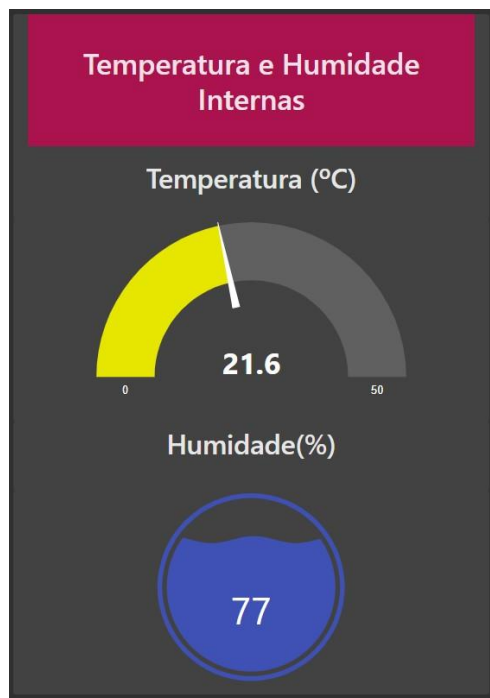
```

Esse código HTML cria um container que exibe a concentração de CO2. A cor de fundo do container é definida com base no valor do CO2, sendo verde para valores abaixo de 800, amarelo para valores entre 800 e 1200, laranja para valores entre 1200 e 1800, e vermelho para valores acima de 1800.

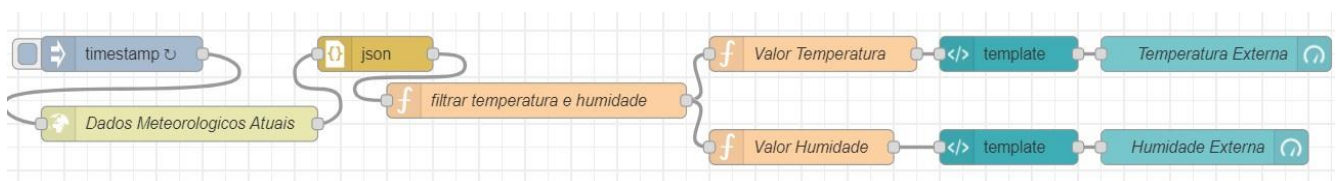
Resultado no dashboard para a qualidade do ar (Concentração de CO2):



Os valores de temperatura e humidades internas são mostrados no nó "ui-gauge", como mostra a seguinte figura:



4. Temperatura e Humidades externas.



Neste passo, para obter os valores externos de temperatura e humidade, é utilizado a API do OpenWeather. Esse serviço fornece informações meteorológicas atuais para uma determinada localização, que é definida por meio de coordenadas de latitude e longitude.

O nó "http request" é responsável por fazer uma solicitação à API do OpenWeather usando o URL

(<https://api.openweathermap.org/data/2.5/weather?lat=41.8076&lon=-6.7606&units=metric&appid=chaveAcesso>).

Esse URL contém as coordenadas da posição geográfica desejada, bem como os parâmetros "units=metric" para retornar os valores de temperatura em Celsius.

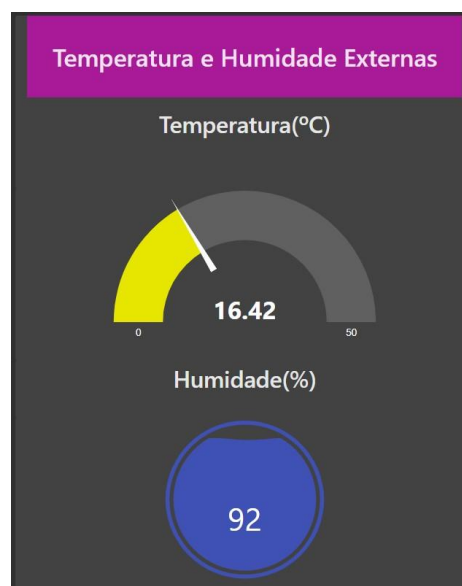
Após a solicitação ser feita, os dados retornados são convertidos em um objeto JSON, para facilidade de acesso às propriedades.

Como apenas os valores de temperatura e humidade são necessários, o nó de função é utilizado para filtrar esses valores e criar um novo objeto com essas informações.

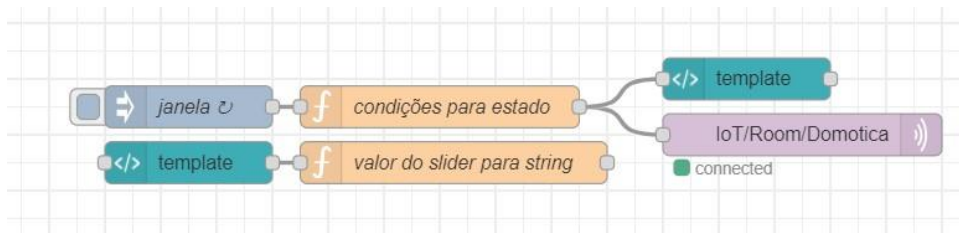
```
1 var temperature = msg.payload.main.temp;
2
3 var externalData = {
4   temperatura: temperature,
5   humidade : msg.payload.main.humidity,
6 }
7
8 msg.payload = externalData;
9
10 return msg;
```

Esse novo objeto é atribuído à propriedade "msg.payload" para que possa ser encaminhado para o restante do fluxo.

De seguida, o fluxo é desdobrado em 2, para os dados de temperatura e humidade serem encaminhados para nó ui-gauge do node-red dashboard. Pelo caminho através do nó ui-template é criado um container, onde os dados serão mostrados, e aplicadas formatações no título “Temperatura e Humidade Externas”. Resultado final no dashboard é dado pela seguinte figura:



5. Posições janela e da cortina.



Para o valor do estado da janela e da cortina, são usadas os dados recolhidos, tanto pelos sensores do sistema wokwi, que simula o ambiente interno, como os dados externos da API do OpenWeather e do IPMA.

No caso da janela, ela pode ser definida em 3 posições (aberta, semi-aberta e fechada) e são usados os dados da concentração de co2 interna, temperatura e humidades externas, o estado atual definido pelo utilizador, bem como id da intensidade da precipitação obtido do IPMA para o dia atual (Previsão Meteorológica Diária até 5 dias agregada por Local).

Estes valores foram previamente definidos em variáveis globais no node-red e que agora são acedidos para criar a lógica e as relações entre os dados para definir o estado final da janela.

Como domótica envolve o controlo localmente ou remotamente, foi criado um slider no ui-template, em html, css e js para o utilizador poder definir o estado da janela. O valor definido pelo utilizador envolve 4 modos (fechada, semi-aberta, aberta e automático), e tem maior relevância sobre os dados recolhidos.

No nó de função, são referenciados os valores das variáveis globais e é criado um objeto global chamado window_state para armazenar informações sobre o estado da janela e sobre os dados atuais.


```

var co2 = global.get("co2");
var idPrec = global.get("id_precipitacao");
var temp_ext = global.get("temp_ext");
var hum_ext = global.get("hum_ext");
var manual_window = global.get("Windowstr");

const window_state = {};

window_state.window = "fechado";
window_state.co2 = co2;
window_state.temp_ext = temp_ext;
window_state.hum_ext = hum_ext;
window_state.id_prec = idPrec;

```

É utilizado um bloco switch-case com base na intensidade da precipitação (idPrec) para definir o estado da janela.

A variável idPrec que se refere à intensidade de precipitação para o dia atual tem maior relevância sobre todos os dados recolhidos, uma vez que não é muito recomendado ter a janela aberta quando chove, e é definido em 0 – Sem precipitação, 1-Precipitação Fraca, 2-Precipitação moderada e 3-Precipitação forte. Valores de 3 e 4 a janela é automaticamente definida no estado fechada, podendo ser alterada se o utilizador definir manualmente

```

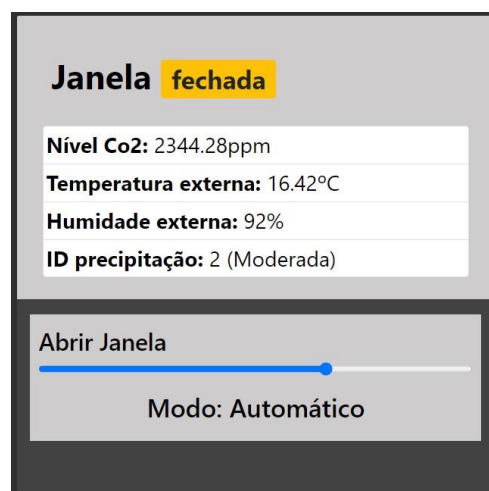
switch(idPrec){
    //id precipitação do dia
    case 0:
        if (manual_window == "automatico") {
            if(co2>=1800 && temp_ext>20 && hum_ext<=60){
                window_state.window = "aberta";
            } else if ((co2>=1200 && co2<=1800) && ( temp_ext>13 &&temp_ext < 21) && hum_ext <=20){
                window_state.window = "semi-aberta";
            } else if ((co2 >= 1200 && co2 <= 1800) && (temp_ext < 13) && hum_ext >= 40){
                window_state.window = "fechada";
            }
        }else{
            window_state.window = manual_window;
        }
        break;
    case 1:
        if (manual_window == "automatico") {
            if ((co2>=1200 && co2<=1800) && temp_ext > 15 && hum_ext <= 40) {
                window_state.window = "semi-aberta";
            } else if (co2 < 1200 && temp_ext < 13 && hum_ext >= 70) {
                window_state.window = "fechada";
            }
        }else{
            window_state.window= manual_window;
        }
        break;
}

```

Para os restantes valores (0 e 1), se o utilizador definir o estado automático, o estado é definido com base nos valores de CO2, temperatura externa e humidade externa, definido como "aberta", "semi-aberta" ou "fechada". Também é verificado se o modo da janela é "automático" ou definido manualmente pelo utilizador.

O estado da janela é publicado no tópico IoT/Room/Domotica, que o sistema interno do wokwi subscreve e é acionado o servo Motor para o simular o estado da janela.

No dashboard, o utilizador pode interagir com a janela, movendo o slider, ou apenas monitorar o estado.



Para a cortina, a ideia é a mesma, mas neste caso os dados relevantes serão de temperatura (temp_int) e humidade (hum_int) internas, o valor de luminosidade (lux) obtido do sistema wowki, e o estado da cortina definido pelo utilizador (cortinastr). É criado um novo objeto c_state para armazenar os valores correntes dos sensores e o estado da cortina para ser mostrado no nó ui-template.

```

var temp_int = global.get("temp_int");
var hum_int = global.get("hum_int");
var lux = global.get("sensor_lux");
var cortinastr = global.get("cortinastr");

var c_state = {};

c_state.cortina = "baixa";
c_state.temp_int = temp_int;
c_state.hum_int = hum_int;
c_state.lux = lux;

```

A cortina terá 3 estados (baixa, média, alta) e o dado recolhido mais relevante e preferencial para o seu estado será o de sensor da medida da iluminação, obtido do wokwi. Como exemplo, a próxima figura mostra os estados que a cortina pode tomar:

```

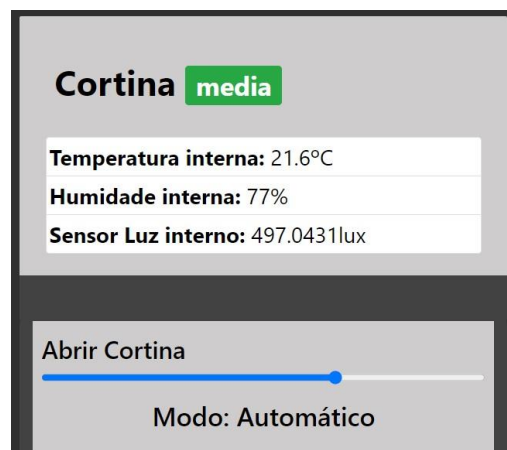
switch(true){
  case lux<=1000:
    if (cortinastr == "automatico") {
      if(temp_int < 15 && hum_int > 20){
        c_state.cortina = "alta";
      } else if (temp_int >15 && hum_int > 20){
        c_state.cortina = "media";
      } else if (temp_int >= 25 && hum_int < 20) {
        c_state.cortina = "baixa";
      }
    }else{
      c_state.cortina = cortinastr;
    }
    break;
  case (lux>=1000 &&lux<=10000):
    if (cortinastr == "automatico") {
      if(temp_int >=15 && hum_int<20)
        c_state.cortina = "media";
    }else{
      c_state.cortina = cortinastr;
    }
    break;
}

```

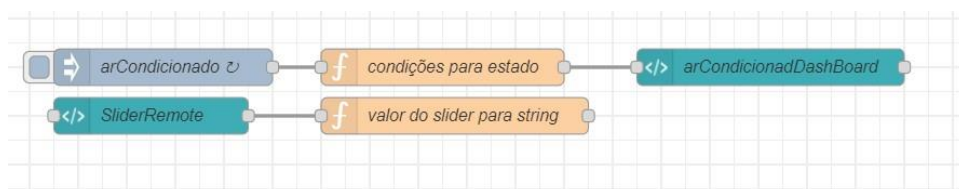
Se a variável lux for menor ou igual a 1000, são verificadas as condições relacionadas ao estado da cortina com base nos valores de temperatura interna e humidade interna. Dependendo dessas condições, o estado da cortina é definido como "alta", "média" ou "baixa". Também é verificado se o modo da cortina é "automático" ou definido manualmente pelo utilizador.

Se lux estiver entre 1000 e 10000, é verificada a condição para o estado da cortina com base no valor de temperatura e humidade interna. Dependendo dessa condição, o estado da cortina é definido como "média". Também é verificado se o modo da cortina é "automático" ou definido manualmente. Após a definição do estado da cortina, o valor é armazenado na variável global "cortina" e o objeto c_state é atribuído à propriedade msg.payload para ser encaminhado para o resto do fluxo.

No dashboard, o resultado final é demonstrado na seguinte figura:



6. Modo do ar condicionado



No modo do ar-condicionado, podem ser definidos 4 estados (desligado, ventilando, aquecendo e arrefecendo).

O código do nó de função começa obtendo os valores das variáveis globais e cria um objeto chamado aC_state. Esse objeto armazena informações sobre o estado do ar condicionado, como o estado atual do ar condicionado (ac), a concentração de CO2 (co2), a temperatura interna (temp_int), a umidade interna (hum_int) e o estado da janela (window). Também é definido o valor que o utilizador definiu (manualAC).

```
aC_state.hum_int = hum_int;  
aC_state.window = windowsState;
```

O bloco switch-case é utilizado para definir o estado do ar condicionado com base no estado da janela (window):

Exemplo de uma condição para definir o modo do ar condicionado:

```
switch (windowsState) {  
    // aberta  
    case "aberta":  
        if (manualAC == "automatico") {  
            if (co2 <= 1800 && temp_int > 15 && hum_int <= 20) {  
                aC_state.ac = "desligado";  
            }  
        } else {  
            aC_state.ac = manualAC;  
        }  
        break;  
}
```

Se o estado da janela for "aberta", são verificadas as condições relacionadas ao estado do ar condicionado com base nos valores de CO2, temperatura interna e humidade interna.

Dependendo dessas condições, o estado do ar condicionado é definido como "desligado", uma vez que a janela aberta e o ar-condicionado a funcionar iria ter um gasto desnecessário de energia. É verificado se o modo do ar condicionado é "automático" ou definido manualmente pelo utilizador, tendo sempre a preferencia sobre a análise dos dados recolhidos.

7. Controlo da iluminação (LED)

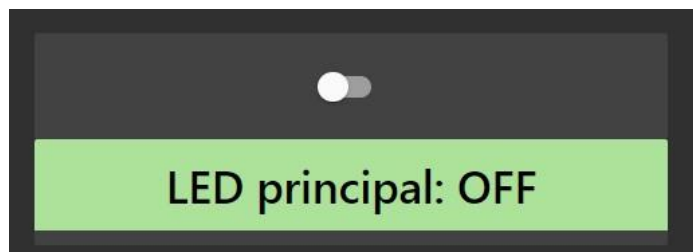
O utilizador, tal como nas interações com a janela, cortina e ar condicionado, pode definir o estado de uma lâmpada LED instalada no ambiente.



Para isso usou-se o nó switch para definir o estado da lâmpada, seguido do nó de função, onde esse estado é guardado em um objeto a ser publicado no tópico `lot/Room/Domotica`, subscrito pelo sistema wokwi, no ambiente da sala de estar instalado.

```
1  var json = {};  
2  
3  json.led_state = msg.payload;  
4  msg.payload = json;  
5  
6  return msg;  
7
```

Interação com o LED no dashBoard:



Conclusão:

Neste projeto, é desenvolvido um sistema domótico para controle e automação de um ambiente residencial, visando proporcionar

conforto, eficiência energética e praticidade aos utilizadores do ambiente.

O sistema integra diversos dispositivos, sensores e serviços externos para gerenciar de forma inteligente as condições ambientais, como temperatura, humidade, luminosidade, estado da janela e da cortina, além de oferecer controle remoto e automático do sistema de ar condicionado.

Durante a implementação, foi utilizado o Node-RED como plataforma de desenvolvimento, que nos permite criar um fluxo de dados e definir a lógica de controle de maneira visual e intuitiva.

Integramos sensores simulados do ambiente interno, obtendo informações precisas sobre temperatura e humidade, além de dados externos fornecidos por serviços como OpenWeather e IPMA, que nos ofereceram informações meteorológicas atualizadas da localização específica.

A janela e a cortina foram controladas com base em uma série de condições, levando em consideração a concentração de CO₂, temperatura e humidade externas, bem como a intensidade da precipitação.

Essas condições foram previamente definidas e armazenadas em variáveis globais, permitindo que o sistema adaptasse o estado da janela de forma automatizada ou manual, dependendo das preferências do utilizador.

Da mesma forma, o sistema de ar condicionado foi integrado ao fluxo de automação, levando em conta as condições ambientais internas e o estado da janela. Com base nessas informações, o sistema determinou o modo de operação mais adequado, como desligado, ventilando, aquecendo ou arrefecendo.

Mais uma vez, os utilizadores têm a opção de definir manualmente o modo de operação do ar condicionado ou deixar

que o sistema tomasse decisões automáticas.

Ao finalizar o projeto, observa-se que a aplicação da domótica proporciona uma experiência mais agradável, ao mesmo tempo em que promove a economia de energia e a sustentabilidade ambiental. Através do controle remoto e automático dos dispositivos, é possível otimizar o uso de recursos, adaptando-os às condições ambientais e às necessidades dos utilizadores.

Em resumo, o projeto de automação residencial desenvolvido demonstrou a viabilidade e os benefícios da aplicação da domótica. Com a integração inteligente de dispositivos, sensores e serviços externos, foi possível criar um ambiente confortável, eficiente e de fácil gerenciamento.

O sistema domótico desenvolvido apresenta grande potencial para futuras aplicações, contribuindo para a melhoria da qualidade de vida em geral.