Smart Home: Janela Inteligente

Lucas de Camargo Souza*
Prof. Wyllian Bezerra da Silva
Universidade Federal de Santa Catarina (UFSC)
EMB5636 Projeto Integrador I

***Resumo*—Com o objetivo de intensificar os conhecimentos em temas relevantes como *Internet of Things* e sistemas embarcados, este trabalho explora suas aplicações por meio de uma rede *smart home* utilizando dispositivos de baixo custo. Esta rede consiste basicamente de uma janela residencial que é operada através de um motor conectado a um *gateway*. O sistema é capaz de abrir e fechar a janela baseando-se em dados de sensoriamento meteorológico, como um simples sensor de chuva ou dados de servidores de terceiros. Os dispositivos deste projeto são desenvolvidos utilizando microcontroladores do tipo ESP8266 e a comunicação entre os diferentes sistemas da rede é implementada por meio do o protocolo MQTT. O custo desta implementação gira em torno de R$ 290,00 e o tempo de implementação, em torno de oito semanas.**

***Palavras-chave*—Smart Home, IoT, Mecatrônica, MQTT, ESP8266, NodeMCU, Sistemas Embarcados.**

I. Introdução

A evolução da digitalização possibilita a administração automatizada de recursos, que antes eram operados manualmente tendo o ser humano como fator de decisão. O levantamento e armazenamento inteligente de dados trouxe aplicações nas indústrias, que as fazem passar por um novo processo de modernização, este designado como Indústria 4.0 [1, 2]. O fato de a digitalização estar fortemente ligada à computação e que há disponibilidade de dispositivos de sensoriamento de baixo custo, faz com que seja possível a sua implementação por usuários comuns e estudantes de engenharia.

O termo *Internet of Things*, ou IoT (traduzido: "Internet das Coisas"), descreve a crescente interligação entre objetos inteligentes como também a conexão destes com a Internet. Diferentes destes objetos, sendo eles do dia a dia ou algo como máquinas industriais, são desenvolvidos com processadores e sensores embarcados, de forma que são capazes de se comunicarem através de seus endereços IPs (*Internet Protocol*) [3]. Designa-se por *smart home* uma aplicação de IoT, em que os métodos e sistemas tecnológicos são aplicados a edifícios e residências, de forma que o ponto principal é o aumento do bem-estar e da qualidade de vida [4].

Dada a necessidade de manter ambientes arejados como forma de melhorar a circulação e qualidade do ar e, consequentemente, a qualidade de vida, muitas pessoas investigam a possibilidade de deixar as janelas abertas ao sair de casa, tendo como fator determinante a previsão climática do dia e a segurança. Excluindo-se a abordagem deste último fator, como por exemplo no caso em que a pessoa mora em um apartamento e a probabilidade de ocorrer invasão domiciliar pela janela é baixa, tem-se que a decisão de deixar a janela aberta ou fechada ao sair de casa é apenas baseada nas condições climáticas do dia, como temperatura externa, intensidade dos ventos e precipitação. Este trabalho busca investigar e desenvolver soluções baseadas nos conceitos apresentados acima, que automatizam a abertura ou fechamento de uma ou mais janelas residenciais em função de certas condições meteorológicas, sendo a ocorrência de chuva a principal delas.

A Seção II apresenta tanto o conceito geral de *smart home* e seus atributos, quanto uma discussão sobre a sua aplicação. Da mesma forma, na seção III é introduzido o funcionamento do protocolo de mensagem MQTT. Uma revisão literária de trabalhos relacionados e fundamentações teóricas é apresentada em IV. Os métodos de execução do projeto são descritos em V. A descrição das implementações dos códigos estão descritas na seção Seção VI. Os resultados das implementações bem como discussões são apresentados na Seção VII. A Seção VIII apresenta os materiais necessários para a execução deste projeto bem como seus respectivos custos. Os apêndices no final disponibilizam todo o código implementado, bem como seu funcionamento, adaptado do repositório GitHub oficial deste projeto, disponível em [5].

II. Smart Home

Disponível em [4], a organização alemã Verbraucherzentrale define o funcionamento de uma *smart home* pelos seguintes elementos:

- dispositivos finais, chamados de **Autores**, que estão de certa forma relacionados com um controle inteligente, como climatizadores, lâmpadas, cortinas, janelas e eletrodomésticos em geral;
- **Dispositivos de Entrada**, como displays *touch*, controladores de temperatura, tablets e smartphones. Outro método de entrada muito comum no mercado é dado por comandos de voz. Por esses dispositivos é possível definir valores desejados como a temperatura em uma sala;
- **Sensores**, que são indispensáveis em um sistema de controle inteligente. Por meio destes é possível definir se para alcançar a temperatura desejada em uma sala é mais eficiente abrir janelas e portas ou ativar um sistema de climatização;
- uma central de unidade de controle, denominada como ***Gateway***, para a qual os dados de sensoriamento e comandos serão enviados. O *gateway* possibilita a troca de informações entre os elementos da rede de *smart home*;

Correspondência ao autor: lucas_camargo@hotmail.com.br matrícula UFSC: 16205041

- uma **Rede de Comunicação**, responsável por integrar os autores e sensores ao *gateway*. A conexão pode ser estabelecida via cabo ou rádio, por exemplo.

Embora muito se diz sobre princípios em que uma *smart home* tenha como objetivo tornar o consumo residencial de energia mais eficiente, uma publicação na revista Nature mostra que esta percepção ainda não pode ser devidamente comprovada [6], como também mostrado em [7]. Logo, é provável que a implementação de uma automação residencial aumente o uso de energia elétrica. Ainda, esta mesma pesquisa discute que os grandes fornecedores de automação residencial prometem por meio de seus produtos melhorar as interações sociais, porém é importante notar que automações e monitoramentos residenciais podem trazer impactos relacionados à convivência familiar, como por exemplo o uso de câmeras, sensores de presença, etc. como meio de perseguir e manipular membros da residência, como esposas, dado que grande parte dos entusiastas desta tecnologia é ainda tipicamente formada por homens. Conclui-se então que a pesquisa e desenvolvimento deste tipo de tecnologia deve investigar como os dispositivos *smart* irão agir no uso geral da energia elétrica dentro das casa e também levantar os impactos positivos e negativos destas aplicações nas interações sociais, além de na segurança e no bem-estar.

III. MQTT

O protocolo de troca de mensagens MQTT (*Message Queue Telemetry Transportation*) foi introduzido em 1999 e teve sua licença liberada gratuitamente a partir da versão 3.1, em 2010 pela empresa IBM. O MQTT foi desenvolvido para ser leve, para funcionar com dispositivos com consumo de energia restrito e baixa largura de banda, utilizado para comunicação M2M (*Machine-to-Machine*), em sistemas IoT que funcionam através da conexão TCP/IP (*Transmission Control Protocol/Internet Protocol*). Em 2016, o protocolo foi regularizado como o padrão ISO/IEC 20922:2016 [8]. O fato do MQTT operar por meio do protocolo TCP/IP, que é utilizado por qualquer sistema ou equipamento com conexão à Internet, faz com que o MQTT possa ser implementado sem qualquer outro custo e módulo suplementar além do Wi-Fi.

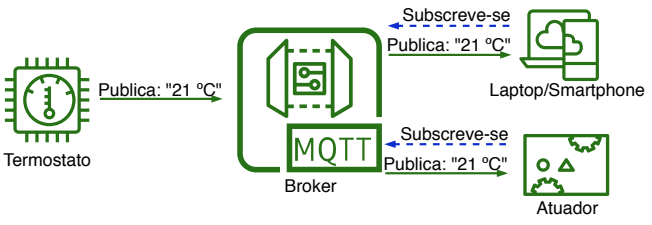

Figura 1. Representação do funcionamento da arquitetura *publish/subscribe* do protocolo de comunicação MQTT.

O Funcionamento do MQTT é descrito por um cliente que publica mensagens para um servidor, denominado *broker* neste contexto, que são então redirecionadas a outros clientes que estão subscritos ao tópico daquela mensagem, ou ainda, a mensagem pode ficar retida no *broker* para subscrições futuras. O tópico é caracterizado pelo endereço em qual a mensagem é publicada. Estes tópicos são estruturados da mesma forma que diretórios em um sistema operacional. Os clientes podem se subscrever para diferentes tópicos e receberem toda mensagem publicada em cada um destes tópicos [9]. Este tipo de arquitetura é denominado *publish/subscribe* e está representado na Figura 1. Para garantir que uma mensagem é enviada ao broker, o protocolo MQTT define três níveis de qualidade de serviço, ou QoS (*Quality of Service*) [10]:

- QoS 0 - a mensagem é enviada no máximo uma vez, sem a notificação de que ela foi devidamente recebida. É utilizado em conexões com fio ou por dispositivos muito restringidos;
- QoS 1 - a mensagem é enviada pelo menos uma vez e, para cada mensagem enviada, uma notificação de reconhecimento, ou *acknowledgement*, é enviada de volta para o emissor. Caso o emissor não seja notificado em um tempo específico, ele então envia a mensagem novamente;
- QoS 2 - a mensagem é enviada exatamente uma vez. Este nível está acima do QoS 1 e garante que uma mensagem duplicada não seja enviada para o cliente receptor. É utilizado em aplicações críticas.

Uma vantagem do MQTT é que uma mensagem pode ser constituída em qualquer formato. Logo, um cliente emissor e um cliente receptor devem antes estabelecer qual será o formato dos dados.

IV. Revisão Literária

A. Fundamentação Teórica

Naik discute em [9] critérios de escolha de um protocolo efetivo para sistemas IoT, sendo os protocolos MQTT, CoAP (*Constrained Application Protocol*), AMQP (*Advanced Message Queuing Protocol*) e HTTP (*Hypertext Transfer Protocol*). A respeito do MQTT, o artigo o coloca como um protocolo de mensagens para aplicações leves, com redes largas de pequenos dispositivos que precisam ser monitorados ou controlados de um servidor. Não é desenvolvido para transferências ponto a ponto e nem para *multicast* de dados para muitos receptores. É um protocolo bem básico, de baixa complexidade e que oferece apenas algumas opções de controle. Entretanto, o fato de o CoAP operar por conexões UDP (*User Datagram Protocol*) ou SCTP (*Stream Control Transmission Protocol*) faz com o seu uso seja mais restrito em redes domésticas de Internet em relação ao MQTT, que utiliza conexão TCP. O texto então propõe uma análise comparativa e relativa com os outros protocolos. Em geral, o MQTT tem resultados parecidos com o CoAP para aplicações IoT, que também é um protocolo desenvolvido, em 2010, para utilização em pequenos dispositivos, interoperando com juntamente com o HTTP e a RESTful Web (*Representational State Transfer*) através de *proxies* simples. Mais especificamente, o MQTT obtém bons resultados quando se trata do tamanho da mensagem, pois é um protocolo binário e que normalmente requer um *header* fixo de 2 bytes com *payloads* de mensagens de até 256 MB. Entretanto, o fato de utilizar conexão TCP aumenta o *overhead* de mensagem e portanto o seu tamanho em relação ao CoAP. No quesito de consumo de energia, o MQTT possui vantagens por ter sido desenvolvido para baixas larguras de banda e para utilização em dispositivos com poucos recursos, como

microcontroladores de 8 bits. Porém, devido à conexão TCP, o MQTT consome mais largura de banda do que o CoAP para transferir uma mesma quantidade de dados. O ponto em que o MQTT se sobressai como o melhor protocolo é na questão da confiabilidade e interoperabilidade, pois este protocolo oferece três níveis de QoS e, ainda, a conexão TCP beneficia a garantia de entrega de mensagem. Outro ponto vantajoso é o seu grande uso em comunicações do tipo M2M. Pois o MQTT tem sido aplicado pelo maior número de organizações, apesar de não possuir nenhuma padronização global. É um protocolo utilizado por grandes organizações como IBM, Facebook, Eurotech, Cisco, Red Hat e Amazon Web Services (AWS). Por fim, o MQTT é um protocolo emergente para IoT e de código livre.

B. Trabalhos Relacionados

Tiwari e Matta discutem em [11] estabelecer uma arquitetura eficiente para uma rede *smart home*. Neste artigo também é estabelecida uma forma de gerenciar recursos em IoT com base em diferentes camadas de implementação. A arquitetura aqui proposta é dada pelo mapeamento de dispositivos inteligentes com base em sua localização dentro de uma residência, como por exemplo a ação de ligar a televisão deve ser mapeada para uma instrução que percorrerá o endereço deste aparelho definido pelo cômodo onde ele se encontra, no caso, na sala. Além disso, são propostas interfaces de comandos para alguns dispositivos.

O artigo [12] mostra a aplicação de um serviço IoT para controlar e monitorar a temperatura de um cômodo, como também acionar um alarme de incêndio e conter o fogo. Para isso foi utilizado o protocolo de troca de mensagens MQTT, que comunica com o serviço comercial de *smart home* AWS. Os dispositivos inteligentes foram feitos utilizando o microcontrolador *Arduino* conectado à rede Wi-Fi, que por sua vez se comunica com o *broker* do MQTT implementado em nuvem no serviço AWS. Nesta aplicação é possível determinar uma temperatura desejada dentro de uma sala, de forma que através dos dados do sensor é possível decidir de forma autônoma se o ar-condicionado deve ou não ser ligado. Por fim, o uso deste serviço pode ser feito via smartphone.

O artigo [13] propõe implementar um sistema de *smart grid* baseado em IoT que monitora o uso de energia elétrica de dispositivos domésticos, como lâmpadas. Os dados são obtidos utilizando o microcontrolador *Arduino*, juntamente com o microcontrolador e módulo Wi-Fi ESP8266 [14], e enviados para um *broker* utilizando o protocolo de troca de mensagens MQTT. Esses dados são por fins armazenados e disponibilizados por acesso via smartphone. O usuário pode então manter um controle de seu gasto de energia elétrica e também realizar o pagamento da conta utilizando RFID (*Radio-Frequency Identification*).

V. MATERIAIS E MÉTODOS

O aluno deverá trabalhar em cada dos elementos de uma *smart home*, como descritos em II. A aplicação desta *smart home* será feita em duas fases: desenvolvimento e integração de uma rede de *smart home*; e a implementação de uma janela automática inteligente a esta rede. Neste projeto, a janela inteligente será o único sistema presente na rede. Este sistema é composto por dois elementos fundamentais: uma janela eletrônica (ou eletronicamente adaptada), que define um autor do sistema, e um ou mais sensores, também podendo ser fontes de informações como a temperatura atual fornecida por um servidor de previsão climática. Esta janela deverá se abrir ou fechar dependendo das condições climáticas como presença de chuva e vento. O usuário poderá também exigir a abertura ou fechamento da janela via um método de entrada, como por exemplo uma interface web. Sensores de chuva poderão ser utilizados para coletar informações relevantes para a decisão de operação da janela. É também importante destacar que este sistema poderá ser aplicado para mais janelas presentes em uma residência.

Considerando ainda uma primeira fase de planejamento e seleção de recursos, estabelece-se um **pré-levantamento de requisitos** que possibilite uma primeira visão geral para um primeiro desenvolvimento do projeto:

1) Cada sistema da rede é implementado em um microcontrolador do tipo ESP8266 [13, 15];
2) A comunicação entre os autores e sensores com o sistema ocorre via MQTT [9, 10, 16];
3) O ambiente deve dispor de uma rede Wi-Fi;
4) A janela deve ser "de correr"e sem travas;
5) A janela abre e fecha automaticamente;
6) A janela pode ser operada manualmente;
7) A posição da janela é binária: aberta ou fechada.

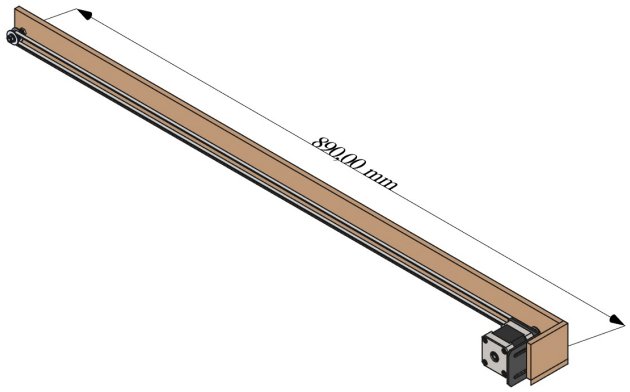

Figura 2. Ilustração de um modelo de atuador para uma janela de correr.

O requisito 4 estabelece que o atuador do sistema deve operar um movimento guiado linearmente que seja capaz de empurrar e puxar a janela sobre o trilho. Logo, determina-se como solução o uso de um motor de passo do tipo Nema 17 atrelado a uma correia, formando uma guia linear. Esta guia pode ser facilmente implementada utilizando uma correia acoplada ao eixo do motor, tal que o seu movimento linear seja capaz de "puxar", ou "arrastar", a janela sobre o trilho. A conexão entre a correia e a janela pode ser feita utilizando um fio. Este modelo está representado na Figura 2. A dimensão de 890,00 mm possibilita a instalação do atuador em diferentes tamanhos de janela. O controle do motor e, respectivamente, da janela é feito utilizando um *drive* para motor de passo,

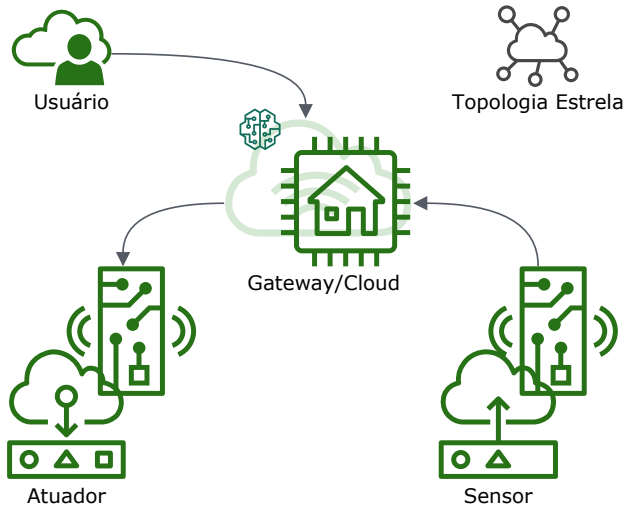

Figura 3. Arquitetura da rede *smart home* proposta em topologia estrela.

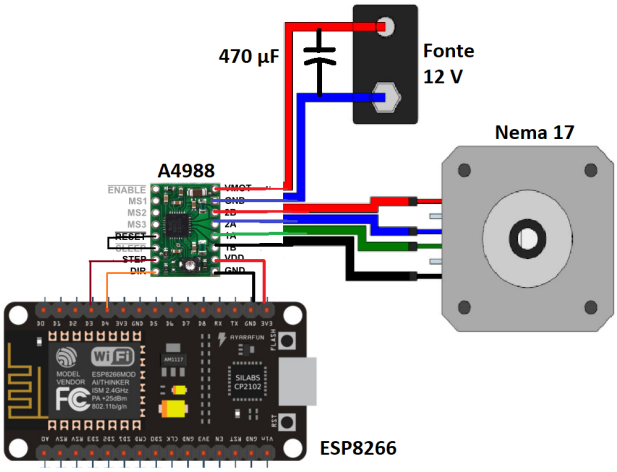

Figura 4. Modelo do circuito eletrônico para controlar o motor Nema 17 através do microcontrolador ESP8266.

modelo A4988 [17], tal que este *drive* é conectado a um microcontrolador ESP8266. Parte da programação do software está, então, aplicada ao gerenciamento do motor elétrico.

A princípio o método geral de implementação resume-se no uso de três sistemas representando a rede *smart home*, cada um destes implementado em um dispositivo microcontrolador ESP8266. O dispositivo principal trata-se de um *gateway*, por onde as informações irão fluir e também onde ocorrerá o processamento de informações e tomadas de decisão. É neste sistema que estará implementado o *broker* do protocolo de troca de mensagens MQTT (v. requisito 2). O segundo dispositivo representará um autor da rede, sendo este o atuador da janela. Por fim, o terceiro dispositivo será um sensor meteorológico, que enviará dados periodicamente para a central. Esta arquitetura configura uma topologia estrela, em que há uma central a qual todos os dispositivos estão conectados e a comunicação entre dispositivos só é possível por meio desta central, aqui chamada de *gateway*. Ainda, a central é o único meio de comunicação com a rede WAN (*Wide Area Network)*. As vantagens desta topologia são consistência, rapidez e facilidade de implementação e depuração, contra a desvantagem trazida pelo meio de comunicação sem fio, pois todos os dispositivos devem ter conexão com a central, o quê está em função da qualidade do sinal Wi-Fi e suas interferências [18]. Este modelo proposto está representado na Figura 3 com um atuador e um sensor, respectivamente uma janela e um sensor meteorológico.

VI. Implementação

Começou-se pela montagem do circuito eletrônico do motor de passo Nema 17, juntamente com o ESP8266 e o drive A4988 do motor. Um esquemático do circuito eletrônico básico está representado na Figura 4.

A implementação mecânica refere-se ao modelo dado pela Figura 2. Para isso, foi utilizado como base da guia linear um pedaço de madeira reciclada, retirada de paletes, tal que o corte foi feito utilizando uma serra circular. Ainda, utilizou-se uma correia, tensores e polias do tipo GT2 para formar a guia.

O projeto do software iniciou-se pelo desenvolvimento de uma biblioteca para o microcontrolador ESP8266 que disponibilize os métodos básicos de um atuador de janela, que se resumem em duas funções principais: abrir e fechar. Para isso, foi utilizada a biblioteca *AccelStepper*, disponível em [19], e a partir dela foram feitas classes herdeiras. Um diagrama simplificado está representado na Figura 5. Neste modelo, é necessário configurar alguns parâmetros referentes à janela e ao atuador, como a distância a ser percorrida pela guia para abrir/fechar a janela, o raio do eixo da guia linear e a velocidade máxima e aceleração do motor de passo.

A. Teste preliminar

O primeiro teste executado constituiu-se em conectar os três projetos mecânico, eletrônico e de software. Foi feito então um pequeno algoritmo para o microcontrolador ESP8266 que utiliza as funções da classe *SmartWindow*, representada na Figura 5. O código disponibiliza como parâmetros de entrada abrir ou fechar a janela com uma certa velocidade de pico e aceleração do motor de passo. Por exemplo, digita-se como entrada *serial* para o microcontrolador dados como: abrir janela, 1080 °/s, 120 °/s^2. A seguir, a guia foi posicionada junto à janela e tentou-se puxar a janela sobre o trilho a partir de um fio de nylon acoplado à correia e ao vidro da janela através de um prendedor de sucção. Uma ilustração do modelo é dada na Figura 6.

B. Testes definitivos

O segundo teste envolveu o desenvolvimento de uma interface para a comunicação MQTT entre os dispositivos da rede. Para facilitar a implementação, foi feita uma classe `Logger` utilizada para enviar mensagens de informação, aviso e erro. Nesta classe é possível definir o nível de registro das mensagens e enviá-las via comunicação serial e/ou MQTT. Quando enviadas por MQTT, essas mensagens são destinadas a um tópico específico com endereço `.../log`, que podem então ser acessadas por outros clientes. Isso faz com que a

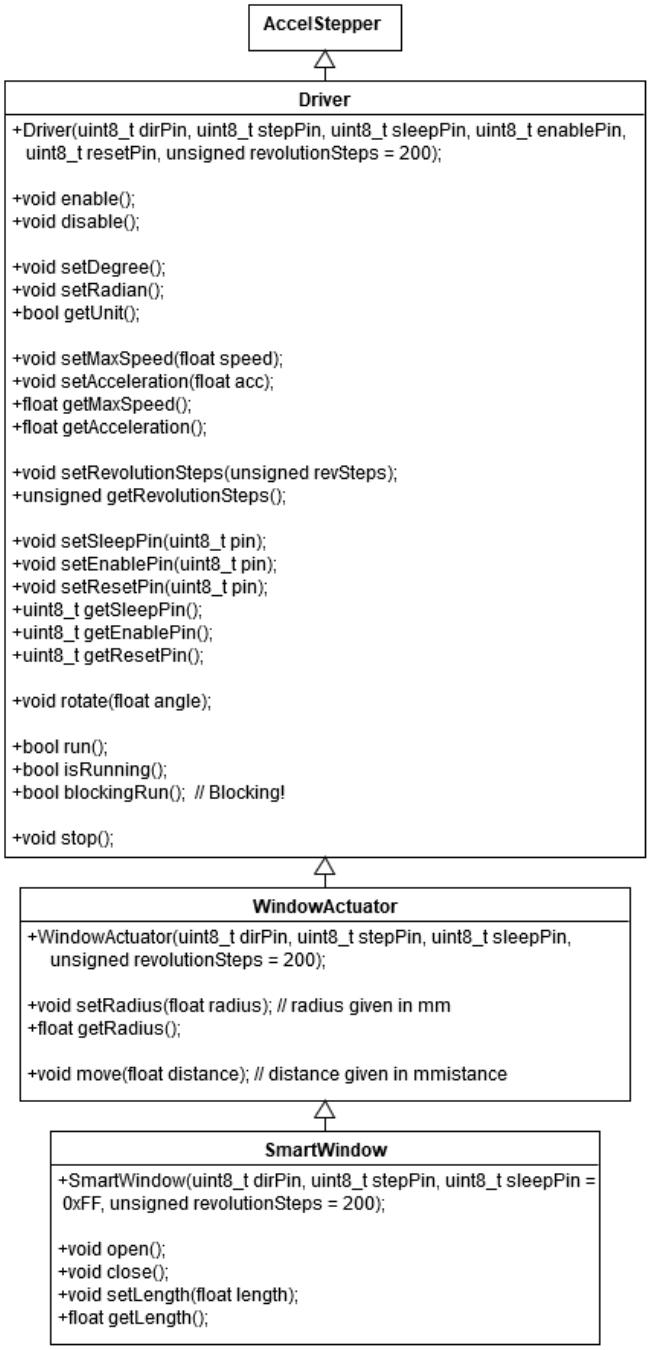

Figura 5. Diagrama de classes simplificado da biblioteca utilizada para controlar o atuador da janela.

troca de mensagens de depuração não esteja apenas retida à comunicação serial, que também é apenas utilizada quando preciso. Ainda, utilizando a biblioteca `NTPClient` [20]. Desta forma, arquitetou-se o primeiro dispositivo, o atuador da janela, como capaz de abrir e fechar a janela e receber e enviar parâmetros de configuração no formato JSON (*JavaScript Object Notation*). Esta aplicação foi feita utilizando a biblioteca `PubSubClient` [21], para o cliente MQTT e a biblioteca `ArduinoJson` [22] para o gerenciamento de dados de tipo JSON. A princípio, na aplicação é definido o nome e senha

Figura 6. Modelo da implementação do teste preliminar.

da rede local de Internet e os dados de conexão com o *broker* do MQTT, como IP e porta. Todos os outros parâmetros de configuração podem ser alterados através via MQTT. Um tópico fixo foi criado com o objetivo de sempre possibilitar a alteração do tópico base do atuador, a verificação deste tópico e a reconfiguração do dispositivo para as configurações padrões. Esta estrutura é dada como:

- `SWALPHA01`
  - `/topic/read`: recebe como argumento o tópico de resposta; publica no tópico de resposta o tópico base atual.
  - `/topic/write`: recebe como argumento um novo tópico base.
  - `/reset`: reconfigura o dispositivo com as configurações padrões.

Por exemplo, para configurar um novo tópico base, envia-se a mensagem `smarthome/window` para o tópico `SWALPHA01/topic/write`. Para verificar o tópico base, o cliente deve estar subscrito a um tópico de resposta, como por exemplo `smarthome/window/response` e enviá-lo como mensagem para o tópico `SWALPHA01`.

A partir do tópico base, tem-se então a estrutura de tópicos principal:

- `/log`: tópico em que as mensagens de depuração são publicadas.
- `/open`: comando para abrir a janela.
- `/close`: comando para fechar a janela.
- `/config/read`: recebe como argumento o tópico de resposta; publica no tópico de resposta uma mensagem do tipo JSON com os parâmetros de configurações e seus valores atuais.
- `/config/write`: recebe como argumento um ou mais parâmetros de configuração em formato JSON.
- `/config/save`: salva as configurações atuais na memória não volátil.

- `/config/load`: recupera da memória não volátil as configurações salvas. Essas configurações já são sempre recuperadas na inicialização do microcontrolador.
- `/config/reset`: reconfigura o dispositivo com as configurações padrões.

Abaixo estão definidos os parâmetros de configuração do atuador da janela bem como seus tipos de dados e valores padrões:

```
// GRPIO da porta DIR do driver A4988
uint8_t dirPin = 4;
// GRPIO da porta STEP do driver A4988
uint8_t stepPin = 5;
// GRPIO da porta SLEEP do driver A4988
uint8_t slpPin = 16;
// Se true, inverte a direcao positiva de rotacao do
    ↪ motor
bool inverted = false;
// Numero de passos para giro de 360 graus do motor
unsigned revSteps = 200;
// Raio do conjunto polia-correia
float radius = 6.35943935;
// Comprimento da guia do atuador
float length = 500;
// Velocidade maxima do motor em grau/s
float maxSpeed = 1080;
// Aceleracao do motor em grau/s2
float acc = 360;
//GPIO do sensor de fim de curso da posicao aberta
uint8_t limOpenSwitch = 0;
//GPIO do sensor de fim de curso da posicao fechada
uint8_t limCloseSwitch = 1;
// UTC local (Brasil: -3)
int8_t timeUTC = -3;
// Se true, envia dados pela comunicacao serial
bool serialOutput = true;
// Topico base MQTT. DEVICE_ID_MAX_LENGTH=128
char mqttTopicRoot[DEVICE_ID_MAX_LENGTH] = "
    ↪ SWALPHA01";
// Nivel de registro de mensagens de depuracao
uint8_t logLevel = Logger::LOG_LEVEL_INFO;
```

O segundo dispositivo implementado trata-se do *broker* do MQTT. Para esta implementação foi utilizada a biblioteca `uMQTTBroker` [23]. Esta biblioteca possui algumas restrições, como por exemplo o único nível de QoS suportado é o zero. Tais restrições ocorrem devido ao baixo poder de processamento do microcontrolador ESP8266. Entretanto, é o suficiente para a execução de um servidor como este, tornando-o uma ótima alternativa ao normalmente utilizado Raspberry Pi [24], pois é muito mais barato.

O terceiro e último dispositivo é a estação climática. Ao invés de utilizar sensores para a coleta de dados, optou-se por obter os dados meteorológicos do servidor OpenWeather [25], pois a quantidade de dados é maior e mais precisa, além de fornecer a previsão climática durante o dia. Estes dados são obtidos do servidor a partir de uma requisição HTTP, que os retorna em formato JSON. Estes dados são então tratados na aplicação e, a partir deles, é formado um novo conjunto de dados também em formato JSON, contendo informações climáticas atuais e previstas, que incluem clima (nublado, chuvoso, ensolarado, etc.), temperatura, sensação térmica, umidade e velocidade dos ventos. Esse grande conjunto de dados e previsões facilitarão o algoritmo de tomada de decisões para quando abrir ou fechar a janela. Sua estrutura de tópicos é dada por um conjunto de atributos com métodos *get* e *set* via MQTT. Por exemplo, para configurar a cidade de onde os dados serão obtidos, escreve-se o nome da cidade, mais a sigla do país (sem espaços), *e.g.* `Joinville,BR`, no tópico `city/set`. Para a leitura da cidade atual, publica-se um tópico de resposta no tópico `city/get`. Os outros atributos são definidos da mesma forma. Além dos tópicos dos atributos listados abaixo, também tem-se `/save` e `/load`, da mesma forma que para o atuador da janela.

- `/city`: `<cidade,sígla>`, a localidade de onde os dados meteorológicos serão obtidos.
- `/npredictions`: quantidade de previsões climáticas.
- `/topic`: tópico base da aplicação.
- `/apiKey`: chave de acesso para a requisição de dados do servidor OpenWeather.
- `/period`: período de tempo, em segundos, em que a aplicação enviará dados ao *broker*.

Outra implementação realizada é a do dispositivo de entrada para dados e comandos relativos à aplicação. Para isso, utilizou-se em um tablet Android o aplicativo MQTT Dashboard [26], em que foi configurado o endereço do *broker* e os tópicos dos dispositivos. Este aplicativo exibe o controle e configurações do atuador, além das informações meteorológicas atuais.

Por último, executa-se estes dois clientes ao *broker* simultaneamente e, utilizando o software MQTT Spy [27], verifica-se e realiza-se a depuração do funcionamento dos clientes mais o *broker*. Com este software é possível subscrever-se a diferentes tópicos e também publicas mensagens. O próximo passo será a implementação do dispositivo de tomada de decisões que possibilitará finalmente abrir e fechar a janela em função dos dados meteorológicos fornecidos.

C. Janela automatizada

Após a implementação dos três principais serviços que compõem a rede smart home, respectivamente o *broker*, o atuador e o sensor, executa-se o desenvolvimento do serviço tomador de decisões, tal que a classe é denominada `AutomatedWindow`. Esta classe subscreve-se ao serviço da estação meteorológica, recebendo informações sobre as condições climáticas atuais e algumas previsões, e então publica para o serviço da janela inteligente, executando a operação de abrir ou fechar a janela em função das preferências configuradas pelo usuário. Essas configurações determinam as condições para que a janela esteja **aberta**. Caso pelo menos uma das condições não coincida, a janela é então fechada. Esses parâmetros estão determinados abaixo:

- Identificação do clima, dado pelo identificador correspondente à definição da página OpenWeather [25], sendo os identificadores relacionados à quantidade de nuvens no céu.
- Intervalo de temperatura, dado em graus Celsius.
- Velocidade máxima dos ventos, dada em metros por segundo.
- Umidade do ar, dada em porcentagem.
- Número de previsões climáticas a serem consideradas na tomada de decisão, por exemplo, caso a próxima previsão não coincida com as condições de janela aberta, então a janela já é fechada.

Ainda, é possível ativar e desativar o automatizador. A interface deste serviço é também toda dada através da comunicação MQTT por tópicos do tipo *get* e *set*.

Por fim, todas as implementações, bem como suas descrições adaptadas do repositório do GitHub, disponível em [5], estão listadas nos apêndices.

VII. Resultados e Discussões

O teste preliminar foi um sucesso quanto ao controle do motor e à operação do movimento da correia guia. O software possibilitou fazer testes para diferentes velocidade de rotação do motor e obteve-se que uma rotação ideal está entre 360 e 1080 °/s. Para a aceleração, um bom valor estaria entre 90 e 240 °/s^2. Entretanto, o acoplamento da guia à janela não obteve sucesso. Aconteceu que a força exercida pela correia na polia fez com ela deslizasse sobre o eixo do motor. Este deslizamento pode ser proveniente do tamanho dos dentes da polia do eixo do motor. O desafio é vencer a força de atrito entre a janela e o trilho para tirá-la da inércia e colocá-la em movimento. O problema é que a janela é arrastada sobre o trilho sem o uso de roldanas ou rolamentos. Uma possível solução seria então diminuir o atrito utilizando um óleo lubrificante ou também utilizar uma polia com mais dentes para evitar o arrastamento da correia. A solução ideal seria utilizar roldanas e rolamentos no trilho da janela, porém isso necessitaria de um reprojeto do desenho da janela e portanto é inviável. As maiores dificuldades encontradas no desenvolvimento do projeto estão relacionadas à parte mecânica, pois a confecção de mecanismos exige ferramentas e materiais de custos elevados e, então, há a exigência que o projeto mecânico seja o mais simplificado possível.

O teste definitivo foi um sucesso em todas as etapas. Tudo ocorreu como esperado e as aplicações funcionaram devidamente. Não houve problemas em relação ao *broker* e suas limitações no ESP8266. Utilizando o dispositivo de entrada no aplicativo MQTT Dashboard, representado na Figura 7, foi possível abrir e fechar a janela além de configurar diferentes parâmetros de operação. Ainda, os dados meteorológicos são periodicamente enviados ao *broker* e também mostrados no aplicativo. Utilizando o MQTT Spy, é possível rastrear o fluxo de mensagens entre os dispositivos, além de verificar constantemente o registro de mensagens de depuração fornecidas pelos dispositivos. A Figura 8 mostra algumas mensagens registradas pelo MQTT Spy durante a execução simultânea das aplicações.

A implementação do serviço de tomada de decisões para abrir e fechar a janela resultou no modelo final do projeto. Foi decidido incorporar este serviço à mesma aplicação do *broker*, porém ele poderia estar também incorporado na estação climática ou até mesmo no próprio serviço da janela. O encapsulamento deste serviço foi então fundamental para permitir futuras alterações. Durante sua implementação foi considerado para um serviço `AutomatedWindow` apenas uma janela, porém seria mais interessante configurar uma lista de janelas para o serviço, que gerenciaria um grupo de janelas baseado nas preferências climáticas. Uma outra fase interessante do projeto, seria revisar os códigos e criar uma padronização, possibilitando mais compartilhamento de classes

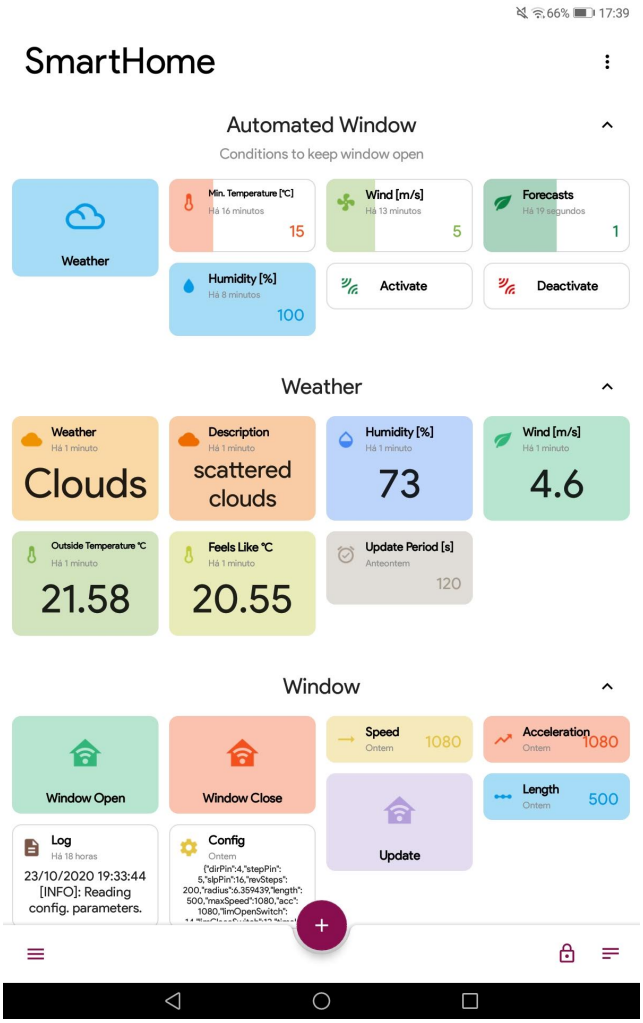

Figura 7. Aplicação desenvolvida no aplicativo MQTT Dashboard para Android [26].

e menos repetições de código, como por exemplo uma classe base para a criação de serviços. Por fim, a Figura 7 mostra o aplicativo MQTT Dashboard com todos os serviços integrados, finalizando este projeto.

VIII. Materiais e Custos

Dado os requisitos estabelecidos até agora e a ideia do projeto, tem-se na Tabela I a listagem de materiais necessários e seus respectivos custos.

Além dos materiais básicos necessários para a implementação, necessita-se também de ferramentas que possibilitem e facilitem o desenvolvimento do projeto. Em geral, para qualquer trabalho com eletrônica, recomenda-se a ter à disposição equipamentos básicos como multímetro, ferro de solda, *protoboard* etc.

IX. Conclusão e Contribuição

Implementar uma aplicação de *smart home* tem como objetivo trazer ao estudante de engenharia a proximidade com temas recorrentes como IoT, aplicação web, digitalização e indústria 4.0. Desta forma, este projeto terá como fins didáticos

Tabela I
RELAÇÃO PROPOSTA DE MATERIAIS NECESSÁRIOS E CUSTOS.

Materiais necessários			
Descrição	Quantidade	Valor Unitário	Total
Motor de Passo Nema 17HS44001 4 Kgf	1	80,00	80,00
Drive p/ Motor de Passo A4988	2	13,00	26,00
Fonte Chaveada 12V 2A	1	20,00	20,00
Conector P4 Fêmea Borne	4	2,00	8,00
Suporte L p/ Motor Nema	1	20,00	20,00
Regulador de Tensão CI 7805	3	4,50	10,00
NodeMCU WiFi V3 Modul ESP8266 ESP-12E	3	26,10	78,30
Capacitor Elet. 47uF 16V	5	0,20	1,00
Kit Polias Gt2 + Correia Dentada	1	50,00	50,00
Total			R$ 296,80

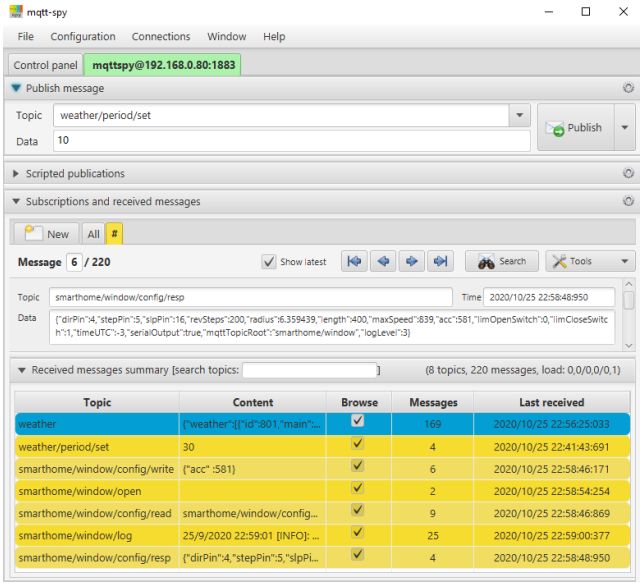

Figura 8. Registro do fluxo de mensagens durante a execução simultânea das aplicações utilizando o software MQTT Spy [27].

o conhecimento das estruturas básicas dos temas mencionados, como também das ferramentas de implementação disponíveis para a comunidade de desenvolvedores de software. Ainda, uma vez familiarizado com os conceitos abordados neste projeto, o aluno será capaz de expandir sua implementação para usos pessoais, profissionais e até comerciais após a entrega do trabalho final.

O desenvolvimento de uma janela residencial inteligente contribuirá para a climatização e qualidade do ar de ambientes residenciais na ausência de pessoas para abrir ou fechar a janela com base nas situações climáticas externas, como dias chuvosos, ventosos e ensolarados. Esta contribuição tem grande importância para a crise sanitária em tempos de epidemias, em que ambientes arejados são essenciais. Dada a recorrente necessidade de levantar possíveis impactos nas relações pessoais causados por produtos de automação residencial, procura-se aqui o desenvolvimento de um projeto de *smart home* que não tenha como consequência algo que exija a abordagem de questões éticas e sociais, limitando-se apenas às áreas da engenharia.

Por fim, o domínio dos temas da era da digitalização é de alto importância para a engenharia mecatrônica no cenário moderno, dando visivelmente destaque ao profissional com tal capacitação.

REFERÊNCIAS

[1] "Maschinenmarkt - Was ist die Industrie 4.0?" [Online]. Available: https://www.maschinenmarkt.vogel.de/was-ist-industrie-40-smarte-technologien-der-zukunft-a-624620/

[2] "Deutsches Bundesministerium für Wirtschaft und Energie - Internet der Dinge." [Online]. Available: https://www.bmwi.de/Redaktion/DE/Artikel/Digitale-Welt/internet-der-dinge.html

[3] "Lexikon - Internet of Things | Gründerszene.de," https://www.gruenderszene.de/lexikon/begriffe/internet-of-things, (Acessado em 15/09/2020).

[4] "Smart Home - Das "intelligente Zuhause" | Verbraucherzentrale.de," https://www.verbraucherzentrale.de/wissen/umwelt-haushalt/wohnen/smart-home-das-intelligente-zuhause-6882, (Acessado em 15/09/2020).

[5] L. de Camargo Souza, "Smart home." [Online]. Available: https://github.com/lucasdecamargo/smart-home

[6] L. Nicholls, Y. Strengers, and J. Sadowski, "Social impacts and control in the smart home," *Nature Energy*, vol. 5, no. 3, pp. 180–182, Mar 2020. [Online]. Available: https://doi.org/10.1038/s41560-020-0574-0

[7] T. Hargreaves, C. Wilson, and R. Hauxwell-Baldwin, "Learning to live in a smart home," *Building Research & Information*, vol. 46, no. 1, pp. 127–139, 2018. [Online]. Available: https://doi.org/10.1080/09613218.2017.1286882

[8] "The standard for iot messaging." [Online]. Available: https://mqtt.org/

[9] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *2017 IEEE International Systems Engineering Symposium (ISSE)*, 2017, pp. 1–7.

[10] A. Cornel - Cristian, T. Gabriel, M. Arhip-Calin, and A. Zamfirescu, "Smart home automation with mqtt," in *2019 54th International Universities Power Engineering Conference (UPEC)*, 2019, pp. 1–5.

[11] U. K. Tiwari and P. Matta, "Efficient smart-home architecture: An application of internet of things,"

SSRN Electronic Journal, 2019. [Online]. Available: https://doi.org/10.2139/ssrn.3350330

[12] D. Kang, M. Park, H. Kim, D. Kim, S. Kim, H. Son, and S. Lee, "Room temperature control and fire alarm/-suppression iot service using mqtt on aws," in *2017 International Conference on Platform Technology and Service (PlatCon)*, 2017, pp. 1–5.

[13] M. Kashyap, V. Sharma, and N. Gupta, "Taking mqtt and nodemcu to iot: Communication in internet of things," *Procedia Computer Science*, vol. 132, pp. 1611 – 1618, 2018, international Conference on Computational Intelligence and Data Science. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050918308585

[14] "ESP8266 Arduino Core's documentation." [Online]. Available: https://arduino-esp8266.readthedocs.io/

[15] Y. Amri and M. A. Setiawan, "Improving smart home concept with the internet of things concept using RaspberryPi and NodeMCU," *IOP Conference Series: Materials Science and Engineering*, vol. 325, p. 012021, mar 2018. [Online]. Available: https://doi.org/10.1088%2F1757-899x%2F325%2F1%2F012021

[16] M. Kashyap, V. Sharma, and N. Gupta, "Taking mqtt and nodemcu to iot: Communication in internet of things," *Procedia Computer Science*, vol. 132, pp. 1611 – 1618, 2018, international Conference on Computational Intelligence and Data Science. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050918308585

[17] "A4988." [Online]. Available: https://reprap.org/wiki/A4988

[18] M. Pacelle, "3 topologies driving iot networking standards," Apr 2014. [Online]. Available: http://radar.oreilly.com/2014/04/3-topologies-driving-iot-networking-standards.html

[19] "Accelstepper class reference." [Online]. Available: http://www.airspayce.com/mikem/arduino/AccelStepper/classAccelStepper.html#a68942c66e78fb7f7b5f0cdade6eb7f06

[20] "NTPClient library." [Online]. Available: https://github.com/arduino-libraries/NTPClient

[21] "Arduino client for MQTT." [Online]. Available: https://github.com/knolleary/pubsubclient

[22] "Arduino JSON, url=https://arduinojson.org/."

[23] "MQTT Broker library for ESP8266 Arduino." [Online]. Available: MQTTBrokerlibraryforESP8266Arduino

[24] "Raspberry pi foundation." [Online]. Available: https://www.raspberrypi.org/

[25] "OpenWeather global services." [Online]. Available: https://openweathermap.org/

[26] "Mqtt Dashboard - IoT and Node-RED controller." [Online]. Available: https://play.google.com/store/apps/details?id=com.app.vetru.mqttdashboard

[27] "Eclipse Paho | The Eclipse Foundation - MQTT Spy." [Online]. Available: https://www.eclipse.org/paho/index.php?page=components/mqtt-spy/index.php

As seções abaixo foram adaptadas do repositório do GitHub[1].

APÊNDICE A
LOGGER

This is a **static class** implemented as an alternative to the conventional `Arduino.Serial`output method. This class encapsulates output operations such as printing to COM port and logging to the MQTT broker.

A. *logger.h*

```
#ifndef LOGGER_H
#define LOGGER_H

#include <Arduino.h>
#include <NTPClient.h>

template<class T>
class Logger
{
public:
  enum LogLevel
  {
    LOG_LEVEL_SILENT,
    LOG_LEVEL_ERROR,
    LOG_LEVEL_WARNING,
    LOG_LEVEL_INFO
  };

  static void setSerial(Print * serial) {_serial = serial;}
  static void setMQTT(T * mqtt, String topic = "log") {_mqtt = mqtt; _mqtt_topic = topic;}
  static void setNTP(NTPClient * ntp) {_ntp = ntp;}

  static void setPrefix(String pref) { _prefix = pref; }

  static void setLevel(unsigned level) {_level = level;}
  static unsigned getLevel() {return _level;}

  static void error(String str) {return log(String("[ERROR]: ") + str, LOG_LEVEL_ERROR);}
  static void warning(String str) {return log(String("[WARNING]: ") + str, LOG_LEVEL_WARNING);}
  static void info(String str){return log(String("[INFO]: ") + str, LOG_LEVEL_INFO);}

protected:
  static void log(const String str, unsigned logLvl)
  {
    if(logLvl <= _level)
    {
      String output = str;
      if(_ntp)
      {
        unsigned long epochTime = _ntp->getEpochTime();
        struct tm *ptm = gmtime((time_t *)&epochTime);
        output = String(ptm->tm_mday) + "/" + String(ptm->tm_mon) + "/"
          + String(ptm->tm_year + 1900) + " " + _ntp->getFormattedTime() + " "  + _prefix +
              ↪ output;
      }
      else
      {
        output = _prefix + output;
      }

      if(_serial)
        _serial->print(output+"\n");
      if(_mqtt)
      {
        _mqtt->publish(_mqtt_topic.c_str(), output.c_str(), output.length());
      }
```

[1] https://github.com/lucasdecamargo/smart-home

```
    }
  }

private:
  static Print * _serial;
  static T * _mqtt;
  static String _mqtt_topic;
  static NTPClient * _ntp;
  static unsigned _level;
  static String _prefix;
};

template<class T>
Print * Logger<T>::_serial = nullptr;
template<class T>
T * Logger<T>::_mqtt = nullptr;
template<class T>
NTPClient * Logger<T>::_ntp = nullptr;
template<class T>
unsigned Logger<T>::_level = Logger<T>::LOG_LEVEL_WARNING;
template<class T>
String Logger<T>::_mqtt_topic = "";
template<class T>
String Logger<T>::_prefix = "";

#endif
```

APÊNDICE B
BROKER

It just implements the MQTT broker library uMQTTBroker[2]. Mind the connection limitations described in their page! For example, it does not support QoS greater than zero and neither too many clients connected simultaneously.

Configure your connection parameters in `definitions.h`. There you can set your static IP address for the broker as well as your Wi-Fi settings.

This application also implements the Automation Client for the Smart Window. Check out below in Appendix E!

A. *myBroker.h*

```
#ifndef MY_BROKER_H
#define MY_BROKER_H

#include <uMQTTBroker.h>
#include <Logger.h>

class myMQTTBroker: public uMQTTBroker
{
public:
    typedef Logger<myMQTTBroker> Log;

    myMQTTBroker(uint16_t portno=1883, uint16_t max_subscriptions=10000, uint16_t
        ↪ max_retained_topics=30)
        : uMQTTBroker(portno,max_subscriptions,max_retained_topics), callback(nullptr)
    { /* ... */ }

    virtual bool onConnect(IPAddress addr, uint16_t client_count)
    {
      Log::info(addr.toString()+" connected");
      return true;
    }

    virtual bool onAuth(String username, String password)
    {
      Log::info("Username/Password: "+username+"/"+password);
      return true;
    }

    virtual void onData(String topic, const char *data, uint32_t length)
    {
      char data_str[length+1];
      os_memcpy(data_str, data, length);
      data_str[length] = '\0';

      if(callback)
        callback(topic.c_str(),data,length);
    }

    void set_callback(void (*foo)(const char*,const char*,unsigned int))
    {
        callback = foo;
    }

private:
    void (*callback)(const char*,const char*,uint32_t);
};

#endif
```

B. *definitions.h*

```
#ifndef DEFINITIONS_H
#define DEFINITIONS_H

/* ************************************************************************
```

[2]https://github.com/martin-ger/uMQTTBroker

```
 * MAKE CHANGES HERE IF NEEDED!!
 * ********************************************************************** */
// WiFi Settings
#define ssid "myWiFi"
#define psk "myWiFiPassword"
// MQTT Settings
#define mqtt_broker "192.168.0.80"  // Static IP Address
#define mqtt_gateway "192.168.0.1"  // Static Gateway IP Address
#define mqtt_subnet "255.255.255.0" // Subnet
#define mqtt_broker_port 1883
#define mqtt_max_subscriptions 10000
#define mqtt_max_retained_topics 30
/* ********************************************************************** */

#endif
```

C. *Broker.ino*

```
#include <ESP8266WiFi.h>
#include <uMQTTBroker.h>
#include <WiFiUdp.h>
#include <NTPClient.h>
#include <Logger.h>
#include "AutomationClient.h"
#include "myBroker.h"
#include "definitions.h"

myMQTTBroker myBroker(mqtt_broker_port,mqtt_max_subscriptions,mqtt_max_retained_topics);
typedef Logger<myMQTTBroker> Log;

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org");

AutomatedWindow<myMQTTBroker> autoWindow(&myBroker);

void setup_wifi()
{
    // Set your Static IP address
    IPAddress local_IP;
    // Set your Gateway IP address
    IPAddress gateway;
    IPAddress subnet;

  local_IP.fromString(mqtt_broker);
  gateway.fromString(mqtt_gateway);
  subnet.fromString(mqtt_subnet);

    // Configures static IP address
  if (!WiFi.config(local_IP, gateway, subnet)) {
    Log::error("STA Failed to configure!");
  }

    Log::info("Connecting to " + String(ssid));

    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, psk);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Log::info("Trying to connect...");
    }

    timeClient.update();

    Log::info("WiFi Connected!");
    Log::info("IP address: " + WiFi.localIP().toString());
}

void callback(const char* topic, const char* payload, unsigned int length)
{
```

```
  autoWindow.callback(topic,payload,length);
}

void setup()
{
  Serial.begin(115200);
  Log::setLevel(Log::LOG_LEVEL_INFO);
  Log::setSerial(&Serial);
  Log::setPrefix("Broker");

  // Load automated window configs
  autoWindow.load();

  // Start WiFi
  setup_wifi();

  timeClient.begin();
  timeClient.setTimeOffset(3600*-3);
  Log::setNTP(&timeClient);

  // Start the broker
  Log::info("Starting MQTT broker");
  myBroker.init();
  Log::setMQTT(&myBroker,"log");
  Log::info("The MQTT broker is alive!");

  Log::info("Setting up the Automated Window Client...");
  myBroker.set_callback(callback);
  autoWindow.subscribe();
  Log::info("Ready!");
}

void loop()
{
  // do anything here
  delay(1000);
}
```

APÊNDICE C
SMART WINDOW

This service was made for a window actuator that receives commands via MQTT to open, close or setting up operational parameters. Note that here a linear actuator was considered for sliding windows that are not common in many countries.

It cooperates with another service called `AutomatedWindow` that bridges information coming from the weather station to the window through a decision maker that closes or opens the windows based on the external weather or forecast.

Mind changing `definitions.h` before uploading the code to your ESP8266.

A. MQTT API

First note: every `/read` topic receives as argument another topic where the response should be published to. For the configuration JSON format, check out the topic below.

A base topic is always set for pre-configurations. Note that `SWALPHA01` will always be set as configuration topic. If more smart windows are to be connected, make sure to change `DEVICE_ID` under `definitions.h`.

1. `SWALPHA01/topic/read` Returns the current root topic other than `SWALPHA01`.
2. `SWALPHA01/topic/write` Receives the new root topic as argument.
3. `SWALPHA01/reset` Resets all configurations parameters.

After defining the new root topic, the API is given. Consider including the root topic before every command.

1. `/log` Log messages destination.
2. `/open` Opens the window. No parameters needed.
3. `/close` Closes the window. No parameters needed.
4. `/config/read` Returns current configurations in JSON format.
5. `/config/write` Receives new configuration parameters in JSON format. Not all parameters must be set. You can also just write a new acceleration for example. **Note:** changing pins will only take effect after saving configurations and reinitializing the microcontroller.
6. `/config/save` Saves the current configurations in the static memory that are loaded in every initialization.
7. `/config/load` Loads last saved configuration parameters.
8. `/config/reset` Resets all configuration parameters to their default value.

B. Configuration JSON Format

As an example, default parameters are listed below in the JSON format. Units are given in degrees, millimetres and seconds. Speed and acceleration are related to the rotor, for example, acceleration is equal to $360 \circ /s^2$. Limit switches set to zero means that no switch is used for both closing and opening the window. Changing pins as well as the limit switches will only take effect after saving the new configurations and reinitializing the microcontroller.

```
{
    "dirPin":4,
    "stepPin":5,
    "slpPin":16,
    "revSteps":200,
    "radius":6.359439,
    "length":500,
    "maxSpeed":1080,
    "acc":360,
    "limOpenSwitch":0,
    "limCloseSwitch":0,
    "timeUTC":-3,
    "serialOutput":true,
    "mqttTopicRoot":"SWALPHA01",
    "logLevel":3
}
```

C. WindowActuator.h

```
#ifndef WINDOW_ACTUATOR_H
#define WINDOW_ACTUATOR_H

#include <Arduino.h>
#include <AccelStepper.h>
```

```
6
7 class Driver
8 {
9 public:
10
11     const static bool UNIT_DEGREE = 0;
12     const static bool UNIT_RADIAN = 1;
13
14
15     Driver(uint8_t dirPin, uint8_t stepPin, unsigned revolutionSteps = 200);
16     Driver(uint8_t dirPin, uint8_t stepPin, uint8_t sleepPin, unsigned revolutionSteps = 200);
17     Driver(uint8_t dirPin, uint8_t stepPin, uint8_t sleepPin, uint8_t enablePin,
18             uint8_t resetPin, unsigned revolutionSteps = 200);
19
20     // Enable pins and turn on the driver if enablePin is set.
21     void enable();
22     // Disable pin, disables driver and puts it to sleep if pins are set.
23     void disable();
24
25     void setDegree();
26     void setRadian();
27     bool getUnit();
28
29     void setMaxSpeed(float speed);
30     void setAcceleration(float acc);
31     float getMaxSpeed();
32     float getAcceleration();
33
34     void setRevolutionSteps(unsigned revSteps);
35     unsigned getRevolutionSteps();
36
37     void setSleepPin(uint8_t pin);
38     void setEnablePin(uint8_t pin);
39     void setResetPin(uint8_t pin);
40     uint8_t getSleepPin();
41     uint8_t getEnablePin();
42     uint8_t getResetPin();
43
44     void rotate(float angle);
45
46     /// Poll the motor and step it if a step is due, implementing
47   /// accelerations and decelerations to acheive the target position. You must call this as
48   /// frequently as possible, but at least once per minimum step time interval,
49   /// preferably in your main loop. Note that each call to run() will make at most one step,
       ↪ and then only when a step is due,
50   /// based on the current speed and the time since the last step.
51   /// \return true if the motor is still running to the target position.
52     bool run();
53     /// Checks to see if the motor is currently running to a target
54   /// \return true if the speed is not zero or not at the target position
55     bool isRunning();
56     bool blockingRun();     // Blocking!
57
58     void stop();
59
60 private:
61     AccelStepper _driver;
62     unsigned _revolutionSteps;
63     uint8_t _enablePin = 0xFF;
64     uint8_t _resetPin = 0xFF;
65     uint8_t _sleepPin = 0xFF;
66     bool _unit = UNIT_DEGREE;           // false = Degree, true = Radian
67     float _acceleration = 1.8;
68 };
69
70
71 class WindowActuator : public Driver
72 {
73 public:
```

```
    WindowActuator(uint8_t dirPin, uint8_t stepPin, unsigned revolutionSteps = 200);
    WindowActuator(uint8_t dirPin, uint8_t stepPin, uint8_t sleepPin, unsigned revolutionSteps
        ↪ = 200);

    void setRadius(float radius); // radius given in mm
    float getRadius();

    void move(float distance); // distance given in mm

private:
    float _radius = 20;

};

#endif
```

D. *WindowActuator.cpp*

```
#include "WindowActuator.h"

// % Driver

Driver::Driver(uint8_t dirPin, uint8_t stepPin, unsigned revolutionSteps)
  : _driver(AccelStepper::MotorInterfaceType::DRIVER, stepPin, dirPin)
{
  _revolutionSteps = revolutionSteps;
}

Driver::Driver(uint8_t dirPin, uint8_t stepPin, uint8_t sleepPin, unsigned revolutionSteps)
  : Driver(dirPin, stepPin, revolutionSteps)
{
  _sleepPin = sleepPin;
  pinMode(_sleepPin, OUTPUT);
  digitalWrite(_sleepPin, LOW);
}

Driver::Driver(uint8_t dirPin, uint8_t stepPin, uint8_t sleepPin, uint8_t enablePin,
  uint8_t resetPin, unsigned revolutionSteps)
    : Driver(dirPin, stepPin, revolutionSteps)
{
  _enablePin = enablePin;
  _resetPin = resetPin;
  _sleepPin = sleepPin;

  if(_sleepPin != 0xFF)
  {
    pinMode(_sleepPin, OUTPUT);
    digitalWrite(_sleepPin, LOW);
  }
}


void Driver::enable()
{
  _driver.enableOutputs();
  _driver.setPinsInverted(false,false,false);
  _driver.setEnablePin(_enablePin);

  if(_resetPin != 0xFF)
  {
    pinMode(_resetPin, OUTPUT);
    digitalWrite(_resetPin, HIGH);
  }

  if(_sleepPin != 0xFF)
  {
    digitalWrite(_sleepPin, HIGH);
    delay(2);
  }
}
```

```
53
54
55 void Driver::disable()
56 {
57   if(_resetPin != 0xFF)
58     digitalWrite(_resetPin, LOW);
59   if(_sleepPin != 0xFF)
60     digitalWrite(_sleepPin, LOW);
61
62   _driver.disableOutputs();
63 }
64
65
66 void Driver::setDegree()
67 {
68   _unit = UNIT_DEGREE;
69 }
70
71 void Driver::setRadian()
72 {
73   _unit = UNIT_RADIAN;
74 }
75
76 bool Driver::getUnit()
77 {
78   return _unit;
79 }
80
81
82 void Driver::setMaxSpeed(float speed)
83 {
84   if(_unit == UNIT_DEGREE)
85     return _driver.setMaxSpeed(_revolutionSteps/360.0 * speed);
86   else
87     return _driver.setMaxSpeed(_revolutionSteps/(2*PI) * speed);
88 }
89
90
91 void Driver::setAcceleration(float acc)
92 {
93   _acceleration = abs(acc);
94
95   if(_unit == UNIT_DEGREE)
96     return _driver.setAcceleration(_revolutionSteps/360.0 * _acceleration);
97   else
98     return _driver.setAcceleration(_revolutionSteps/(2*PI) * _acceleration);
99 }
100
101
102 float Driver::getMaxSpeed()
103 {
104   return _driver.maxSpeed();
105 }
106
107
108 float Driver::getAcceleration()
109 {
110   return _acceleration;
111 }
112
113
114 void Driver::setRevolutionSteps(unsigned revSteps)
115 {
116   _revolutionSteps = revSteps;
117 }
118
119 unsigned Driver::getRevolutionSteps()
120 {
121   return _revolutionSteps;
```

```
}


void Driver::setSleepPin(uint8_t pin)
{
  _sleepPin = pin;
}

void Driver::setEnablePin(uint8_t pin)
{
  _enablePin = pin;
}

void Driver::setResetPin(uint8_t pin)
{
  _resetPin = pin;
}

uint8_t Driver::getSleepPin()
{
  return _sleepPin;
}

uint8_t Driver::getEnablePin()
{
  return _enablePin;
}

uint8_t Driver::getResetPin()
{
  return _resetPin;
}



void Driver::rotate(float angle)
{
  if(_unit == UNIT_DEGREE)
    return _driver.move(_revolutionSteps/360.0 * angle);
  else
    return _driver.move(_revolutionSteps/(2.0*PI) * angle);
}


bool Driver::blockingRun()  // Blocking
{
  return _driver.runSpeedToPosition();
}



bool Driver::run()
{
  return _driver.run();
}

bool Driver::isRunning()
{
  return _driver.isRunning();
}


void Driver::stop()
{
  return _driver.stop();
}


```

```
// % WindowActuator

WindowActuator::WindowActuator(uint8_t dirPin, uint8_t stepPin, unsigned revolutionSteps)
  : Driver(dirPin, stepPin, revolutionSteps)
{

}

WindowActuator::WindowActuator(uint8_t dirPin, uint8_t stepPin,
  uint8_t sleepPin, unsigned revolutionSteps)
  : Driver(dirPin, stepPin, sleepPin, revolutionSteps)
{

}


void WindowActuator::setRadius(float radius)
{
  _radius = abs(radius);
}

float WindowActuator::getRadius()
{
  return _radius;
}


void WindowActuator::move(float distance)
{
  if(this->getUnit() == UNIT_DEGREE)
  {
    this->setRadian();
    this->rotate(distance*1.0/_radius);
    this->setDegree();
  }
  else
    return this->rotate(distance*1.0/_radius);
}
```

E. SmartWindow.h

```
#ifndef SMART_WINDOW_H
#define SMART_WINDOW_H

#include <Arduino.h>
#include "WindowActuator.h"
#include "definitions.h"

class LimitSwitch
{
public:
    LimitSwitch(uint8_t pin, bool trigState = true);

    bool read();

private:
    uint8_t _pin;
    bool _trigState;
};


class SmartWindow: public WindowActuator
{
public:
    SmartWindow(const struct config_t config);
    SmartWindow(uint8_t dirPin, uint8_t stepPin, uint8_t sleepPin = 0xFF, unsigned
        ↪ revolutionSteps = 200);

    void open();
    void close();
```

```
29
30      bool run();
31
32      enum SensorType {LIMIT_SWITCH};
33      enum Status {IDLE, OPENING, CLOSING};
34
35      void setSensor(LimitSwitch * openSens, LimitSwitch * closeSens);
36      SensorType getSensorType();
37
38      void setLength(float length);
39      float getLength();
40
41      void setConfig(const struct config_t config);
42
43  private:
44      SensorType _sensType;
45      Status _status;
46      LimitSwitch * _limOpenSwitch = nullptr;
47      LimitSwitch * _limCloseSwitch = nullptr;
48      float _length;
49      bool _inverted;
50  };
51
52  #endif
```

F. SmartWindow.cpp

```
1   #include "SmartWindow.h"
2
3   /* CLASS LIMIT SWITCH */
4   LimitSwitch::LimitSwitch(uint8_t pin, bool trigState)
5       : _pin(pin), _trigState(trigState)
6   {
7       pinMode(pin, INPUT);
8   }
9
10  bool LimitSwitch::read()
11  {
12      return(_trigState ? !digitalRead(_pin) : digitalRead(_pin));
13  }
14  /**********************/
15
16  void SmartWindow::setConfig(const struct config_t config)
17  {
18      _length = config.length;
19      _inverted = config.inverted;
20      setRadius(config.radius);
21      setMaxSpeed(config.maxSpeed);
22      setAcceleration(config.acc);
23      setRevolutionSteps(config.revSteps);
24  }
25
26  SmartWindow::SmartWindow(const struct config_t config)
27      : _sensType(LIMIT_SWITCH), _status(IDLE), _length(config.length), _inverted(config.inverted
            ↪ ),
28       WindowActuator(config.dirPin, config.stepPin, config.slpPin, config.revSteps)
29  {
30      setRadius(config.radius);
31      setMaxSpeed(config.maxSpeed);
32      setAcceleration(config.acc);
33  }
34
35  SmartWindow::SmartWindow(uint8_t dirPin, uint8_t stepPin, uint8_t sleepPin, unsigned
        ↪ revolutionSteps)
36      : _sensType(LIMIT_SWITCH), _status(IDLE), _length(0.0), _inverted(false),
37       WindowActuator(dirPin, stepPin, sleepPin, revolutionSteps)
38  {
39  /*...*/
40  }
41
```

```
42
43 void SmartWindow::open()
44 {
45     if(_sensType == LIMIT_SWITCH)
46     {
47         if(_limOpenSwitch != nullptr)
48         {
49             if(_limOpenSwitch->read())
50                 return;
51         }
52
53         move(_inverted ? -1.0*_length : _length);
54         _status =  OPENING;
55     }
56 }
57
58
59 void SmartWindow::close()
60 {
61     if(_sensType == LIMIT_SWITCH)
62     {
63         if(_limCloseSwitch != nullptr)
64         {
65             if(_limCloseSwitch->read())
66                 return;
67         }
68
69         move(_inverted ? _length : -1.0*_length);
70         _status =  CLOSING;
71     }
72 }
73
74
75 bool SmartWindow::run()
76 {
77     if(_sensType == LIMIT_SWITCH && (_limOpenSwitch != nullptr && _limCloseSwitch != nullptr))
78     {
79         switch(_status)
80         {
81             case IDLE:
82             return WindowActuator::run();
83             break;
84
85             case OPENING:
86             if(!_limOpenSwitch->read())
87             {
88                 bool ret = WindowActuator::run();
89                 if(!ret) _status = IDLE;
90                 return ret;
91             }
92             else
93             {
94                 stop();
95                 _status = IDLE;
96                 return WindowActuator::run();
97             }
98             break;
99
100            case CLOSING:
101            if(!_limCloseSwitch->read())
102            {
103                bool ret = WindowActuator::run();
104                if(!ret) _status = IDLE;
105                return ret;
106            }
107            else
108            {
109                stop();
110                _status = IDLE;
```

```
                return WindowActuator::run();
            }
            break;
        }
    }

    else return WindowActuator::run();
}


void SmartWindow::setSensor(LimitSwitch * openSens, LimitSwitch * closeSens)
{
    _limOpenSwitch = openSens;
    _limCloseSwitch = closeSens;
    _sensType = LIMIT_SWITCH;
}


SmartWindow::SensorType SmartWindow::getSensorType()
{
    return _sensType;
}


void SmartWindow::setLength(float length)
{
    _length = length;
}


float SmartWindow::getLength()
{
    return _length;
}
```

G. *definitions.h*

```
#ifndef DEFINITIONS_H
#define DEFINITIONS_H

#include <PubSubClient.h>
#include <Logger.h>

/* ************************************************************************
 * MAKE CHANGES HERE IF NEEDED!!
 * ************************************************************************ */
// WiFi Settings
#define ssid "myWiFi"
#define psk "myWiFiPassword"
// MQTT Settings
#define mqtt_broker "192.168.0.80"
#define mqtt_broker_port 1883
#define mqtt_id "SmartWindow01"  // DO NEVER USE DUPLICATED ID ON BROKER!
#define mqtt_username ""
#define mqtt_password ""
/* ************************************************************************ */



#define DEVICE_ID "SWALPHA01\0"
#define DEVICE_ID_MAX_LENGTH 128
#define CONFIG_SIZE 15

/* THIS ARE THE DEAFULT VALUES! CHANGE IT IF YOU WANT BUT WATCH OUT! */
typedef struct config_t
{
  uint8_t dirPin = 4;
  uint8_t stepPin = 5;
  uint8_t slpPin = 16;
  bool inverted = false;
```

```
34
35   unsigned revSteps = 200;
36
37   float radius = 6.35943935;
38   float length = 500;
39   float maxSpeed = 1080;
40   float acc = 360;
41
42   uint8_t limOpenSwitch = 0;
43   uint8_t limCloseSwitch = 0;
44
45   int8_t timeUTC = -3;
46
47   bool serialOutput = true;
48
49   char mqttTopicRoot[DEVICE_ID_MAX_LENGTH] = DEVICE_ID;
50
51   uint8_t logLevel = Logger<PubSubClient>::LOG_LEVEL_INFO;
52 };
53
54 #endif
```

H. SmarWindow.ino

```
1  #include <ESP8266WiFi.h>
2  #include <PubSubClient.h>
3  #include <NTPClient.h>
4  #include <WiFiUdp.h>
5  #include <ArduinoJson.h>
6  #include <inttypes.h>
7  #include <stdarg.h>
8  #include <EEPROM.h>
9
10 #include "definitions.h"
11 #include "SmartWindow.h"
12 #include <Logger.h>
13
14 typedef Logger<PubSubClient> Log;
15
16 config_t config;
17
18 WiFiClient espClient;
19 PubSubClient mqttClient(espClient);
20
21 WiFiUDP ntpUDP;
22 NTPClient timeClient(ntpUDP, "pool.ntp.org");
23
24 SmartWindow* sWindow = nullptr;
25 LimitSwitch* openSens = nullptr;
26 LimitSwitch* closeSens = nullptr;
27
28 void mqttUpdateTopic()
29 {
30   String mqttTopicRoot = String(config.mqttTopicRoot);
31   Log::setMQTT(&mqttClient,String(mqttTopicRoot + "/log"));
32   mqttClient.subscribe(String(mqttTopicRoot + "/open").c_str());
33   mqttClient.subscribe(String(mqttTopicRoot + "/close").c_str());
34   mqttClient.subscribe(String(mqttTopicRoot + "/config/read").c_str());
35   mqttClient.subscribe(String(mqttTopicRoot + "/config/write").c_str());
36   mqttClient.subscribe(String(mqttTopicRoot + "/config/save").c_str());
37   mqttClient.subscribe(String(mqttTopicRoot + "/config/load").c_str());
38   mqttClient.subscribe(String(mqttTopicRoot + "/config/reset").c_str());
39 }
40
41 String configSerialize(const struct config_t * confObj)
42 {
43   const size_t capacity = JSON_OBJECT_SIZE(CONFIG_SIZE);
44   StaticJsonDocument<capacity> doc;
45
46   doc["dirPin"] = confObj->dirPin;
```

```
  doc["stepPin"] = confObj->stepPin;
  doc["slpPin"] = confObj->slpPin;
  doc["revSteps"] = confObj->revSteps;
  doc["radius"] = confObj->radius;
  doc["length"] = confObj->length;
  doc["maxSpeed"] = confObj->maxSpeed;
  doc["acc"] = confObj->acc;
  doc["limOpenSwitch"] = confObj->limOpenSwitch;
  doc["limCloseSwitch"] = confObj->limCloseSwitch;
  doc["timeUTC"] = confObj->timeUTC;
  doc["serialOutput"] = confObj->serialOutput;
  doc["mqttTopicRoot"] = confObj->mqttTopicRoot;
  doc["logLevel"] = confObj->logLevel;

  String output = "";

  serializeJson(doc, output);

  return output;
}

void configDeserealize(struct config_t * confObj, String str)
{
  const size_t capacity = JSON_OBJECT_SIZE(CONFIG_SIZE);
  StaticJsonDocument<capacity> doc;

  DeserializationError error = deserializeJson(doc, str);
  if (error)
  {
    Log::error("Failed to deserialize payload message. Error code: " + String(error.c_str()));
    return;
  }

  // Pin changes only take effect after reinitializing the microcontroler
  if(doc["dirPin"])
     confObj->dirPin = doc["dirPin"];
  if(doc["stepPin"])
     confObj->stepPin = doc["stepPin"];
  if(doc["slpPin"])
     confObj->slpPin = doc["slpPin"];

  if(doc["revSteps"])
  {
     confObj->revSteps = doc["revSteps"];
     sWindow->setRevolutionSteps(confObj->revSteps);
  }
  if(doc["radius"])
  {
     confObj->radius = doc["radius"];
     sWindow->setRadius(confObj->radius);
  }
  if(doc["length"])
  {
     confObj->length = doc["length"];
     sWindow->setLength(confObj->length);
  }
  if(doc["maxSpeed"])
  {
     confObj->maxSpeed = doc["maxSpeed"];
     sWindow->setMaxSpeed(confObj->maxSpeed);
  }
  if(doc["acc"])
  {
     confObj->acc = doc["acc"];
     sWindow->setAcceleration(confObj->acc);
  }

  if(doc["limOpenSwitch"])
     confObj->limOpenSwitch = doc["limOpenSwitch"];
```

```
  if(doc["limCloseSwitch"])
     confObj->limCloseSwitch = doc["limCloseSwitch"];

  if(doc["timeUTC"])
  {
     confObj->timeUTC = doc["timeUTC"];
     timeClient.setTimeOffset(3600*config.timeUTC);
  }
  if(doc["serialOutput"])
  {
    confObj->serialOutput = doc["serialOutput"];
    if(confObj->serialOutput)
      Log::setSerial(&Serial);
    else
      Log::setSerial(nullptr);
  }
  if(doc["mqttTopicRoot"])
  {
     mqttClient.unsubscribe(String(String(confObj->mqttTopicRoot) + "/#").c_str());
     // confObj->mqttTopicRoot = doc["mqttTopicRoot"];
     strlcpy(confObj->mqttTopicRoot,(const char*)doc["mqttTopicRoot"],DEVICE_ID_MAX_LENGTH);
     mqttUpdateTopic();
  }
  if(doc["logLevel"])
  {
     confObj->logLevel = doc["logLevel"];
     Log::setLevel(confObj->logLevel);
  }
}


void setup() {
    Serial.begin(115200);
    Log::setSerial(&Serial);
    Log::setLevel(Log::LOG_LEVEL_INFO);
    Log::info("Reading data from EEPROM.");

    EEPROM.begin(512);
    EEPROM.get<struct config_t>(0,config);
    // config.limOpenSwitch = 14;
    // config.limCloseSwitch = 12;
    // EEPROM.put<struct config_t>(0,config);
    // EEPROM.commit();

    timeClient.begin();
    timeClient.setTimeOffset(3600*config.timeUTC);
    Log::setNTP(&timeClient);

    setup_wifi();
    mqttClient.setBufferSize(mqttClient. getBufferSize() *3);
    mqttClient.setServer(mqtt_broker, mqtt_broker_port);
    mqttClient.setCallback(mqttCallback);

    Log::info("MQTT Topic: " + String(config.mqttTopicRoot));

    Log::info("Initializing window actuator.");
    sWindow = new SmartWindow(config);

    if(config.limOpenSwitch != 0 && config.limOpenSwitch != 0)
    {
      openSens = new LimitSwitch(config.limOpenSwitch);
      closeSens = new LimitSwitch(config.limCloseSwitch);
      sWindow->setSensor(openSens, closeSens);
    }
}

void setup_wifi() {
    delay(10);
    Log::info("Connecting to " + String(ssid));
```

```
185
186     WiFi.begin(ssid, psk);
187
188     while (WiFi.status() != WL_CONNECTED) {
189         delay(500);
190         Log::info("Trying to connect...");
191     }
192
193     timeClient.update();
194
195     Log::info("WiFi Connected!");
196     Log::info("IP address: " + WiFi.localIP().toString());
197 }
198
199 void mqttCallback(char* topic, byte* payload, unsigned int length) {
200     String stopic = String(topic);
201     String mqttTopicRoot = String(config.mqttTopicRoot);
202     char msg[length+1];
203     for (int i = 0; i < length; i++) {
204         msg[i] = (char)payload[i];
205     }
206     msg[length] = '\0';
207
208     Log::info("Received message [" + stopic + "]: " + String(msg));
209
210     if(stopic == (String(DEVICE_ID) + "/topic/write"))
211     {
212       Log::info("Changing topic via device root topic to: " + String(msg));
213       //config.mqttTopicRoot = String(msg);
214       configDeserealize(&config, "{\"mqttTopicRoot\":\""  + String(msg) + "\"}");
215     }
216     if(stopic == (String(DEVICE_ID) + "/topic/read"))
217     {
218       Log::info("Reading and sending topic via device root topic.");
219       if(!mqttClient.publish(msg, mqttTopicRoot.c_str()))
220         Log::error("Publish error!");
221     }
222     else if(stopic == (String(DEVICE_ID) + "/reset"))
223     {
224       Log::info("Reseting config. parameters.");
225       config = config_t();
226     }
227
228     else if(stopic == (mqttTopicRoot + "/config/read"))
229     {
230       Log::info("Reading config. parameters.");
231       String output = configSerialize(&config);
232       // Log::info(String("Sending output: " + output));
233       if(!mqttClient.publish(msg, output.c_str()))
234         Log::error("Publish error!");
235     }
236     else if(stopic == (mqttTopicRoot + "/config/write"))
237     {
238       Log::info("Writing config. parameters.");
239       configDeserealize(&config, String(msg));
240     }
241     else if(stopic == (mqttTopicRoot + "/config/reset"))
242     {
243       Log::info("Reseting config. parameters.");
244       config = config_t();
245     }
246     else if(stopic == (mqttTopicRoot + "/config/save"))
247     {
248       Log::info("Saving config. parameters.");
249       EEPROM.put<struct config_t>(0,config);
250       EEPROM.commit();
251     }
252     else if(stopic == (mqttTopicRoot + "/config/load"))
253     {
```

```
254       Log::info("Loading config. parameters.");
255       EEPROM.get<struct config_t>(0,config);
256     }
257 
258     else if(stopic == (mqttTopicRoot + "/open"))
259     {
260       Log::info("Opening window.");
261       // OPEN WINDOW // Implementar parÃćmetros de abrir/fechar janela: acc, vel. etc
262       sWindow->open();
263     }
264     else if(stopic == (mqttTopicRoot + "/close"))
265     {
266       Log::info("Closing window.");
267       // CLOSE WINDOW
268       sWindow->close();
269     }
270 
271 
272 
273 
274     if(strcmp(msg,"on")==0){
275         Log::info("Lights on");
276     }
277     else if(strcmp(msg,"off")==0){
278         Log::info("Lights off");
279     }
280 }
281 
282 void mqttReconnect() {
283     while (!mqttClient.connected()) {
284         Log::info("Reconnecting MQTT.");
285         if (!mqttClient.connect(mqtt_id,mqtt_username,mqtt_password)) {
286             Log::error("Failed to connect, rc=" + mqttClient.state());
287             if(WiFi.status() != WL_CONNECTED)
288               Log::warning("WiFi is not connected!");
289             Log::info("Connect retry in 5 seconds.");
290             delay(5000);
291         }
292     }
293     mqttClient.subscribe(String(DEVICE_ID + String("/topic/write")).c_str());
294     mqttClient.subscribe(String(DEVICE_ID + String("/topic/read")).c_str());
295     mqttClient.subscribe(String(DEVICE_ID + String("/reset")).c_str());
296     mqttUpdateTopic();
297     Log::info("MQTT Connected!");
298 }
299 
300 void loop() {
301     if (!mqttClient.connected())
302         mqttReconnect();
303 
304     mqttClient.loop();
305 
306     // Checks if there is some task staked on SmartWindow
307     if(sWindow->isRunning())
308     {
309       Log::info("Starting window operation.");
310       String mqttTopicRoot = String(config.mqttTopicRoot);
311       // Do not listen to commands!
312       mqttClient.unsubscribe(String(mqttTopicRoot + "/open").c_str());
313       mqttClient.unsubscribe(String(mqttTopicRoot + "/close").c_str());
314       sWindow->enable();
315       while(sWindow->run())
316       {
317         yield();
318       }
319       sWindow->disable();
320       // Reenable listening
321       mqttClient.subscribe(String(mqttTopicRoot + "/open").c_str());
322       mqttClient.subscribe(String(mqttTopicRoot + "/close").c_str());
```

```
323         Log::info("Finished window operation.");
324       }
325       // timeClient.update();
326 }
```

APÊNDICE D
WEATHER STATION

Every smart home you think must include some kind of weather station. Weather and forecast are really important decision factors to take actions inside a house. In order to develop a demonstrator model, I wrote this station service that can be attached to other applications within the ESP8266 or even in one single controller.

Weather data are periodically collected from OpenWeather[3] through its API. In order for that to work, you must create an account and generate your free API key for the HTTP requests. Once you have the key, you can set it by using the MQTT API that I will present below.

In short, the weather client is fully defined in `weather.h` and wrapped in the `.ino` file as an application.

Mind changing `definitions.h` before uploading the code to your ESP8266.

Check out this webpage[4] as well!

A. MQTT API

First note: every `/get` topic receives as argument another topic where the response should be published to. For the output JSON format, check out the topic below.

It consists of getters and setters for its parameters as follows. The **standard root topic** is `weather`.

1. `city/get` Returns the current set city from where weather data is collected and its country code as `<city>,<country code>`, *e.g.* `Berlin,DE`.
2. `city/set` Sets city from where weather data is collected and its country code as `<city>,<country code>`, *e.g.* `Berlin,DE`.
3. `npredictions/get` Returns the number of predictions/forecast data.
4. `npredictions/set` Sets the number of predictions/forecast data. Maximum value tested is two.
5. `topic/get` Returns the current root topic.
6. `topic/set` Sets a new root topic.
7. `apiKey/get` Returns the current API key from OpenWeather.
8. `apiKey/set` Sets an API key from OpenWeather. Consider creating an account and generating one. Otherwise you wont get any response from the weather server.
9. `period/get` Returns the current data request period in seconds.
10. `period/set` Sets a new data request period in seconds. Consider that OpenWeather lets you make a maximum of 1,000,000 calls per month with a free account! This implies in 60 calls per minute.
11. `save` Saves current configuration parameters in the static EEPROM memory.
12. `load` Loads last saved configuration parameters from the static EEPROM memory.

B. Output JSON format

The client will periodically publish a JSON data as below for `npredictions=2`.


```
{
    "weather":
    [
        {
            "id": 801,
            "main": "Cloudy",
            "description": "Pigs are falling from sky! They might want to rule this world...",
            "temp": 23.33,
            "feels_like": 25.66,
            "humidity": 50,
            "wind": 0.21,
            "dt": 1603167280
        },

        {
            "id": 600,
            "main": "Raining",
            "description": "It will rain cows!",
            "temp": 33.33,
```


[3] https://openweathermap.org/

[4] https://openweathermap.org/weather-conditions

```
            "feels_like": 45.66,
            "humidity": 50,
            "wind": 0.21,
            "dt": 1603168000
        },

        {
            "id": 600,
            "main": "Cloudy",
            "description": "Aliens will be seen on sky!",
            "temp": 23.33,
            "feels_like": 25.66,
            "humidity": 50,
            "wind": 0.21,
            "dt": 1603169000
        }
    ]
}
```

C. *weather.h*

```
#ifndef WEATHER_H
#define WEATHER_H

#define ARDUINOJSON_USE_LONG_LONG 1
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <EEPROM.h>
#include <Logger.h>

#define CITY_MAX_LENGTH 128
#define TOPIC_MAX_LENGTH 128
#define APIKEY_MAX_LENGTH 64

class Weather
{
public:
    Weather(String apiKey, WiFiClient* wifiClient)
        : _apiKey(apiKey), _wifiClient(wifiClient)
    {
        _err = "";
    }

    String get(String city, unsigned npredictions = 2)
    {
        // Get first current weather data
        String weather = "";
        if(!httpJsonRequest("/data/2.5/weather?q=" + city + "&APPID=" + _apiKey + "&mode=json&
            ↪ units=metric", weather))
            return String(""); // Error occured

        // Get forecast data
        String forecast = "";
        if(npredictions > 0)
        {
            if(!httpJsonRequest("/data/2.5/forecast?q=" + city + "&APPID=" + _apiKey + "&mode=
                ↪ json&units=metric&cnt=" + String(npredictions), forecast))
                return String(""); // Error occured
        }

        // Deserialize current weather data
        DynamicJsonDocument docWeather(JSON_ARRAY_SIZE(1) + JSON_OBJECT_SIZE(1) + 2*
            ↪ JSON_OBJECT_SIZE(2) + JSON_OBJECT_SIZE(4) + JSON_OBJECT_SIZE(5) +
            ↪ JSON_OBJECT_SIZE(6) + JSON_OBJECT_SIZE(13) + 270);
```

```
        deserializeJson(docWeather,weather);

        // Parse to output json
        DynamicJsonDocument docOutput(JSON_ARRAY_SIZE(1+npredictions) + JSON_OBJECT_SIZE(1) +
            ↪ (1+npredictions)*JSON_OBJECT_SIZE(8) + 150 + 80*npredictions);

        JsonArray w = docOutput.createNestedArray("weather");
        JsonObject w0 = w.createNestedObject();
        w0["id"] = docWeather["weather"][0]["id"];
        w0["main"] = docWeather["weather"][0]["main"];
        w0["description"] = docWeather["weather"][0]["description"];
        w0["temp"] = docWeather["main"]["temp"];
        w0["feels_like"] = docWeather["main"]["feels_like"];
        w0["humidity"] = docWeather["main"]["humidity"];
        w0["wind"] = docWeather["wind"]["speed"];
        w0["dt"] = docWeather["dt"];

        if(npredictions > 0)
        {
            // Deserialize forecast data
            const size_t capacity = (npredictions == 1 ? 2 : npredictions)*JSON_ARRAY_SIZE(1) +
                ↪  JSON_ARRAY_SIZE(npredictions) + (2*npredictions)*JSON_OBJECT_SIZE(1) + (1+
                ↪ npredictions)*JSON_OBJECT_SIZE(2)
                                    + npredictions*JSON_OBJECT_SIZE(4) + JSON_OBJECT_SIZE(5) +
                                        ↪ JSON_OBJECT_SIZE(8) + (2*npredictions)*
                                        ↪ JSON_OBJECT_SIZE(9) + 370 + (npredictions-1)*50;
            DynamicJsonDocument docForecast(capacity);
            deserializeJson(docForecast,forecast);

            for(unsigned i = 0; i < npredictions; i++)
            {
                JsonObject wi = w.createNestedObject();
                wi["id"] = docForecast["list"][i]["weather"][0]["id"];
                wi["main"] = docForecast["list"][i]["weather"][0]["main"];
                wi["description"] = docForecast["list"][i]["weather"][0]["description"];
                wi["temp"] = docForecast["list"][i]["main"]["temp"];
                wi["feels_like"] = docForecast["list"][i]["main"]["feels_like"];
                wi["humidity"] = docForecast["list"][i]["main"]["humidity"];
                wi["wind"] = docForecast["list"][i]["wind"]["speed"];
                wi["dt"] = docForecast["list"][i]["dt"];
            }
        }

        String output = "";
        serializeJson(docOutput, output);

        return output;
    }

    String err()
    {
        String ret = _err;
        _err = "";
        return ret;
    }

    void setApiKey(String apiKey) {_apiKey = apiKey;}
    String getApiKey() {return _apiKey;}



protected:
    bool httpJsonRequest(const String url, String &output)
    {
        // close any connection before send a new request to allow wifiClient make connection
            ↪ to server
        _wifiClient->stop();

        if(_wifiClient->connect(_server,80))
```

```
104         {
105             _wifiClient->println("GET " + url + " HTTP/1.1");
106             _wifiClient->println("Host: " + String(_server));
107             _wifiClient->println("User-Agent: ArduinoWiFi/1.1");
108             _wifiClient->println("Connection: close");
109             _wifiClient->println();
110 
111             unsigned long timeout = millis();
112             while(_wifiClient->available() == 0)
113             {
114                 if(millis() - timeout > 5000)
115                 {
116                     _err = "Client timeout (5s).";
117                     _wifiClient->stop();
118                     return false;
119                 }
120             }
121 
122             char c = 0;
123             int jsonend = 0;
124             bool startJson = false;
125             while(_wifiClient->available())
126             {
127                 c = _wifiClient->read();
128 
129                 if(c == '{')
130                 {
131                     startJson = true;
132                     jsonend++;
133                 }
134 
135                 else if(c == '}')
136                     jsonend--;
137 
138                 if(startJson)
139                     output += c;
140 
141                 if(jsonend == 0 && startJson)
142                 {
143                     startJson = false;
144                     return true;
145                 }
146             }
147         }
148 
149         else
150         {
151             _err = "Connection to server has failed.";
152             return false;
153         }
154 
155         _err = "Something went wrong on trying to request json data. Connection to weather host
                ↪  has might broken.";
156         return false;
157     }
158 
159 private:
160     String _apiKey;
161     WiFiClient* _wifiClient;
162     const char* _server = "api.openweathermap.org";
163 
164 protected:
165     String _err;
166 };
167 
168 
169 /* DO NOT use the same WiFiClient instance for both MQTTClient and WeatherClient.
170  * Use two instances e.g.:
171  * WifiClient mqttWiFiClient;
```

```
 * WifiClient httpWiFiClient;
 *
 * PubSubClient mqttClient(mqttWiFiClient);
 * WeatherMQTT weatherMQTT(..., httpWiFiClient); */
template<typename T>
class WeatherMQTT: public Weather
{
public:
    typedef Logger<T> Log;

    typedef struct
    {
        char city[CITY_MAX_LENGTH];
        char mqttTopic[TOPIC_MAX_LENGTH];
        char apiKey[APIKEY_MAX_LENGTH];
        unsigned npredictions;
        unsigned long period;
        unsigned long lastConnectionTime;
    } args_t;

    WeatherMQTT(String apiKey, WiFiClient * wifiClient, T * mqttClient, String mqttTopic = "
        ↪ weather")
        : _wifiClient(wifiClient), _mqttClient(mqttClient), _mqttTopic(mqttTopic), Weather(
            ↪ apiKey, wifiClient)
    {
        // ...
    }

    void setCity(String city) {_city = city;}
    String getCity() {return _city;}

    void setMqttTopic(String topic) {_mqttTopic = topic;}
    String getMqttTopic() {return _mqttTopic;}

    void setPeriod(unsigned long period) {_period = period*1000; _lastConnectionTime = _period
        ↪ ;}
    unsigned long getPeriod() {return _period;}

    // Specifies the number of the n following forecast data
    void setnPredictions(unsigned val) {_npredictions = val;}
    unsigned getnPredictions(void) {return _npredictions;}

    bool subscribe()
    {
        bool ret = true;
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/city/get").c_str());
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/city/set").c_str());
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/npredictions/get").c_str());
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/npredictions/set").c_str());
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/topic/get").c_str());
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/topic/set").c_str());
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/apiKey/get").c_str());
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/apiKey/set").c_str());
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/period/get").c_str());
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/period/set").c_str());
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/save").c_str());
        ret &= _mqttClient->subscribe(String(getMqttTopic() +"/load").c_str());

        if(!ret)
        {
            _err = "Failed to subscribe to weather related topic.";
            Log::error(_err);
        }

        return ret;
    }

    bool unsubscribe()
    {
```

```
238         bool ret = true;
239         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/city/get").c_str());
240         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/city/set").c_str());
241         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/npredictions/get").c_str());
242         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/npredictions/set").c_str());
243         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/topic/get").c_str());
244         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/topic/set").c_str());
245         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/apiKey/get").c_str());
246         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/apiKey/set").c_str());
247         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/period/get").c_str());
248         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/period/set").c_str());
249         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/save").c_str());
250         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/load").c_str());
251 
252         if(!ret)
253         {
254             _err = "Failed to unsubscribe from weather related topics.";
255             Log::error(_err);
256         }
257 
258         return ret;
259     }
260 
261     bool resubscribe()
262     {
263         if(!unsubscribe())
264             return false;
265         return subscribe();
266     }
267 
268     void setEEPROMAddress(int add) {_eepromAdd = add;}
269     int getEEPROMAddress() {return _eepromAdd;}
270 
271 
272     // Returns the last saved address. You still need to begin EEPROM and commit it!
273     // Begin recommended: EEPROM.begin(sizeof(args_t));
274     // To commit (after calling save): EEPROM.commit();
275     int save(int const address)
276     {
277         args_t s;
278         EEPROM.begin(sizeof(args_t));
279 
280         if(EEPROM.length() < sizeof(s))
281         {
282             _err = "EEPROM length is too small. Please initialize it with EEPROM.begin(sizeof(
                    ↪ s_t))";
283             Log::error(_err);
284             return 0;
285         }
286 
287         strlcpy(s.city,(const char*)getCity().c_str(),CITY_MAX_LENGTH);
288         strlcpy(s.mqttTopic,(const char*)getMqttTopic().c_str(),TOPIC_MAX_LENGTH);
289         strlcpy(s.apiKey,(const char*)getApiKey().c_str(),APIKEY_MAX_LENGTH);
290         s.npredictions = getnPredictions();
291         s.period = getPeriod();
292         s.lastConnectionTime = _lastConnectionTime;
293 
294         EEPROM.put<args_t>(address,s);
295         EEPROM.commit();
296         EEPROM.end();
297 
298         return address + sizeof(s) -1;
299     }
300 
301     int save(void)
302     {
303         return save(_eepromAdd);
304     }
305 
```

```
306     bool load(int const address)
307     {
308         args_t s;
309         EEPROM.begin(sizeof(args_t));
310 
311         if(EEPROM.length() < sizeof(s))
312         {
313             _err = "EEPROM length is too small. Please initialize it with EEPROM.begin(sizeof(
                    ↪ s_t))";
314             Log::error(_err);
315             return false;
316         }
317 
318         EEPROM.get<args_t>(address,s);
319         EEPROM.end();
320 
321         setCity(String(s.city));
322         setMqttTopic(String(s.mqttTopic));
323         setApiKey(String(s.apiKey));
324         setnPredictions(s.npredictions);
325         _period = s.period; // setPeriod multiplies it by 1000!
326         _lastConnectionTime = s.lastConnectionTime;
327 
328 
329         return true;
330     }
331 
332     bool load(void)
333     {
334         return load(_eepromAdd);
335     }
336 
337     // Call this method inside your callback functions with raw arguments
338     // returns false in case of error
339     bool callback(String topic, String payload)
340     {
341         if(topic == getMqttTopic() +"/city/get")
342         {
343             if(!_mqttClient->publish(payload.c_str(),getCity().c_str()))
344             {
345                 _err = "Publish error! Could not publish <" + getCity() + "> to topic <" +
                        ↪ payload + ">.";
346                 Log::error(_err);
347                 return false;
348             }
349         }
350         else if(topic == getMqttTopic() +"/city/set")
351             setCity(payload);
352 
353         else if(topic == getMqttTopic() +"/npredictions/get")
354         {
355             if(!_mqttClient->publish(payload.c_str(),String(getnPredictions()).c_str()))
356             {
357                 _err = "Publish error! Could not publish <" + String(getnPredictions()) + "> to
                        ↪  topic <" + payload + ">.";
358                 Log::error(_err);
359                 return false;
360             }
361         }
362         else if(topic == getMqttTopic() +"/npredictions/set")
363             setnPredictions(payload.toInt());
364 
365         else if(topic == getMqttTopic() +"/topic/get")
366         {
367             if(!_mqttClient->publish(payload.c_str(),getMqttTopic().c_str()))
368             {
369                 _err = "Publish error! Could not publish <" + getMqttTopic() + "> to topic <" +
                        ↪  payload + ">.";
370                 Log::error(_err);
```

```
371                 return false;
372             }
373         }
374         else if(topic == getMqttTopic() +"/topic/set")
375         {
376             if(!unsubscribe())
377                 return false;
378 
379             String hold = getMqttTopic();
380 
381             setMqttTopic(payload);
382 
383             if(!subscribe())
384             {
385                 _err = "Given topic is might unvalid.";
386                 Log::error(_err);
387                 setMqttTopic(hold);
388                 if(!subscribe())
389                 {
390                     _err = "Could not resubscribe to MQTT topic. New given topic was discarded.
                        ↪ ";
391                     Log::error(_err);
392                 }
393                 return false;
394             }
395         }
396 
397         else if(topic == getMqttTopic() +"/period/get")
398         {
399             if(!_mqttClient->publish(payload.c_str(),String(getPeriod()).c_str()))
400             {
401                 _err = "Publish error! Could not publish <" + String(getPeriod()) + "> to topic
                        ↪  <" + payload + ">.";
402                 Log::error(_err);
403                 return false;
404             }
405         }
406         else if(topic == getMqttTopic() +"/period/set")
407             setPeriod((unsigned long)payload.toInt());
408 
409         else if(topic == getMqttTopic() +"/apiKey/get")
410         {
411             if(!_mqttClient->publish(payload.c_str(),getApiKey().c_str()))
412             {
413                 _err = "Publish error! Could not publish <" + getApiKey() + "> to topic <" +
                        ↪ payload + ">.";
414                 Log::error(_err);
415                 return false;
416             }
417         }
418         else if(topic == getMqttTopic() +"/apiKey/set")
419             setApiKey(payload);
420 
421         else if(topic == getMqttTopic() +"/save")
422         {
423             if(!save(getEEPROMAddress()))
424                 return false;
425         }
426         else if(topic == getMqttTopic() +"/load")
427         {
428             if(!load(getEEPROMAddress()))
429                 return false;
430         }
431 
432         return true;
433     }
434 
435     bool callback(char* topic, byte* payload, unsigned int length)
436     {
```

```
        String stopic = String(topic);
        char msg[length+1];
        for (int i = 0; i < length; i++) {
            msg[i] = (char)payload[i];
        }
        msg[length] = '\0';

        return this->callback(stopic, String(msg));
    }

    uint16_t minBufferSize() {return _minMqttBuff + _buffSizeInc*_npredictions;}

    // Call this method inside your loop
    bool run()
    {
        if(millis() - _lastConnectionTime > _period)
        {
            _lastConnectionTime = millis();

            String payload = get(_city, _npredictions);

            if(payload == "")
            {
                Log::error(_err);
                return false;
            }

            if(!_mqttClient->publish(_mqttTopic.c_str(), payload.c_str(), true)) // true ->
                ↪ retained
            {
                _err = "Publish failed. Check if the MQTT Client buffer size matches the
                    ↪ minimum required for your given npredictions, given by WeatherClient::
                    ↪ minBufferSize().";
                Log::error(_err);
                return false;
            }
        }

        return true;
    }

private:
    WiFiClient * _wifiClient;
    T * _mqttClient;
    String _city;
    String _mqttTopic;
    unsigned _npredictions = 2;
    unsigned long _period = 60*1000; // 60 seg = 1 min
    unsigned long _lastConnectionTime = 60*1000;
    const uint16_t _minMqttBuff = 280;
    const uint16_t _buffSizeInc = 242;
    int _eepromAdd = 0;
};


#endif
```

D. *definitions.h*

```
#ifndef DEFINITIONS_H
#define DEFINITIONS_H

/* ************************************************************************
 * MAKE CHANGES HERE IF NEEDED!!
 * ************************************************************************ */
// WiFi Settings
#define ssid "myWiFi"
#define psk "myWiFiPassword"
// MQTT Settings
#define mqtt_broker "192.168.0.80"
```

```
#define mqtt_broker_port 1883
#define mqtt_id "WeatherStation"  // DO NEVER USE DUPLICATED ID ON BROKER!
#define mqtt_username ""
#define mqtt_password ""
/* ************************************************************************** */

#endif
```

E. WeatherClient.ino

```
#define ARDUINOJSON_USE_LONG_LONG 1
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <PubSubClient.h>
#include <Logger.h>
#include "weather.h"
#include "definitions.h"


typedef Logger<PubSubClient> Log;

WiFiClient client;
PubSubClient mqttClient(client);
WiFiClient httpClient;

int status = WL_IDLE_STATUS;

// Get an API Key on https://openweathermap.org/
WeatherMQTT<PubSubClient> weatherService("your_API_key_from_OpenWeatherMap", &httpClient, &
    ↪ mqttClient);

void setup() {
  Serial.begin(115200);
  Log::setLevel(Log::LOG_LEVEL_INFO);
  Log::setSerial(&Serial);
  Log::setPrefix("Weather");

  WiFi.begin(ssid,psk);

  Log::info("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  Log::info("WiFi Connected");
  printWiFiStatus();

  weatherService.load();

  mqttClient.setBufferSize(weatherService.minBufferSize());
    mqttClient.setServer(mqtt_broker, mqtt_broker_port);
    mqttClient.setCallback(mqttCallback);
}

void mqttReconnect() {
    while (!mqttClient.connected())
    {
        Log::info("Reconnecting MQTT.");
        if (!mqttClient.connect(mqtt_id,mqtt_username,mqtt_password)) {
            Log::error(String("Failed to connect to MQTT, rc=") + String(mqttClient.state()));
            if(WiFi.status() != WL_CONNECTED)
              Log::error("WiFi is not connected!");
            Log::info("Connect retry in 5 seconds.");
            delay(5000);
        }
    }
    Log::setMQTT(&mqttClient);

    weatherService.subscribe();
    Log::info("MQTT Connected!");
```

```
}

void loop() {
    if (!mqttClient.connected())
      mqttReconnect();

  mqttClient.loop();
  weatherService.run();
}

void mqttCallback(char* topic, byte* payload, unsigned int length)
{
    weatherService.callback(topic,payload,length);
}

// print Wifi status
void printWiFiStatus() {
  // print the SSID of the network you're attached to:
  Log::info(String("SSID: ") + WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Log::info(String("IP Address: ") + ip.toString());

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Log::info(String("Signal strength (RSSI): ") + String(rssi) + " dBm");
}
```

APÊNDICE E
AUTOMATED WINDOW

This service is a decision maker and it subscribes to the weather client and publishes to the smart window depending on the user preferences. It consists of closing/opening the window according to weather conditions. It is encapsulated in `AutomatedClient.h` and therefore can be implemented in other applications as well rather than in the broker specifically.

Note that the user must set the conditions for the window to be **open**! If these conditions do not match, then it will call the operation to close the window.

A. MQTT API

First note: every `/get` topic receives as argument another topic where the response should be published to.
It consists of getters and setters for its parameters as follows. The **standard root topic** is `automatedWindow`.
For the weather ID, check out this webpage[5]!
Intervals are given in **JSON format** like: `{"min": <value>, "max": <value>}`

1. `/wid/get` Returns the current weather ID interval set. Output is JSON.
2. `/wid/set` Sets a new weather ID interval. To avoid letting your window open during a thunderstorm, the interval is limited to [800, 804]. Input is JSON. Default: `{"min": 800, "max": 804}`
3. `/temp/get` Returns the current temperature interval set. Output is JSON.
4. `/temp/set` Sets a new temperature interval. Input is JSON. Default: `{"min": 16, "max": 50}`
5. `/wind/get` Returns the current maximum wind speed condition set in m/s.
6. `/wind/set` Sets a maximum wind speed condition in m/s. Default: `5.5`.
7. `/humidity/get` Returns the current maximum humidity condition set in %.
8. `/humidity/set` Sets a maximum humidity condition in %. Default: `40`.
9. `/forecast/get` Returns the number of forecasts to consider for taking decision. For example, if it is about to rain in, it closes the window before it even starts to rain.
10. `/forecast/set` Sets the number of forecasts to consider for taking decision. Default: `1`.
11. `/topic/get` Returns the current root topic.
12. `/topic/set` Sets a new root topic. Default: `automatedWindow`.
13. `/activate`
    Activates the automation client. Active by default.
14. `/deactivate` Deactivates the automation client. (Watch out!)
15. `/save` Saves current configuration parameters.
16. `/load` Loads last saved configuration parameters.

B. AutomationClient.h

```
#ifndef AUTOMATION_CLIENT_H
#define AUTOMATION_CLIENT_H

#include <uMQTTBroker.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <EEPROM.h>
#include <Logger.h>

#define TOPIC_MAX_LENGTH 128

template<typename T>
class AutomatedWindow
{
public:
    typedef Logger<T> Log;

    // Weather Limiting Conditions
    // These are conditions for window to be OPEN
    typedef struct
    {
        // Interval Conditions
        // Weather ID
        // Chekout: https://openweathermap.org/weather-conditions
```

[5]https://openweathermap.org/weather-conditions

```
        int wid[2] = {WID_MIN, WID_MAX};
        int temp[2] = {16, 50};         // Degree Celcius
        // Maximal Conditions
        float wind = 5.5;           // m/s
        int humidity = 40;          // %

        // If true, uses next i forecasts as dicision factor to
        // close window instead of current weather.
        // It still uses current weather info to open the window.
        uint8_t forecast = 1;
    } wlconditions_t;

    typedef struct
    {
        char mqttTopic[TOPIC_MAX_LENGTH];
        char weatherTopic[TOPIC_MAX_LENGTH];
        bool active = true;
    } config_t;

    static const int WID_MIN = 800;
    static const int WID_MAX = 804;

    AutomatedWindow(T * mqttClient, String mqttTopic = "automatedWindow")
        : _mqttClient(mqttClient), _mqttTopic(mqttTopic) {}

    // wpl: weather data payload
    bool decide(String wpl)
    {
        // Parses string
        const size_t capacity = JSON_ARRAY_SIZE(3) + JSON_OBJECT_SIZE(1) + 3*JSON_OBJECT_SIZE
            ↪ (8) + 310;
        DynamicJsonDocument doc(capacity);

        deserializeJson<String>(doc, wpl);
        JsonArray weather = doc["weather"];

        // Stays true only if all OPEN conditions are matched
        bool match = true;
        for(int i = 0; i < (_wlcond.forecast<=2 ? _wlcond.forecast+1 : 2); i++)
        {
            match &= weather[i]["id"] >= _wlcond.wid[0] && weather[i]["id"] <= _wlcond.wid[1];
            match &= weather[i]["temp"] >= _wlcond.temp[0] && weather[i]["temp"] <= _wlcond.
                ↪ temp[1];
            match &= weather[i]["wind"] <= _wlcond.wind;
            match &= weather[i]["humidity"] <= _wlcond.wind;
        }

        char dummy = '\0';
        if(match)
        {
            // Opens Window
            _mqttClient->publish(String(_windowTopic + "/open").c_str(),&dummy,sizeof(dummy));
        }
        else
        {
            // Closes Window
            _mqttClient->publish(String(_windowTopic + "/close").c_str(),&dummy,sizeof(dummy));
        }
    }

    void setConditions(wlconditions_t & cond) { _wlcond = cond; }
    wlconditions_t getConditions() { return _wlcond; }

    void setMqttTopic(String topic) {_mqttTopic = topic;}
    String getMqttTopic() {return _mqttTopic;}

    void setWeatherTopic(String topic) { _weatherTopic = topic; }
    String getWeatherTopic() { return _weatherTopic; }

```

```
92      void activate() { _active = true; }
93      void deactivate(){ _active = false; }
94      bool active() { return _active; }
95
96      void setEEPROMAddress(int add) {_eepromAdd = add;}
97      int getEEPROMAddress() {return _eepromAdd;}
98
99      bool subscribe(bool a=false)
100     {
101         bool ret = true;
102         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/wid/get").c_str());
103         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/wid/set").c_str());
104         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/temp/get").c_str());
105         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/temp/set").c_str());
106         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/wind/get").c_str());
107         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/wind/set").c_str());
108         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/humidity/get").c_str());
109         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/humidity/set").c_str());
110         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/forecast/get").c_str());
111         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/forecast/set").c_str());
112         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/topic/get").c_str());
113         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/topic/set").c_str());
114         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/activate").c_str());
115         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/deactivate").c_str());
116         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/save").c_str());
117         ret &= _mqttClient->subscribe(String(getMqttTopic() +"/load").c_str());
118         if(_active)
119             ret &= _mqttClient->subscribe(_weatherTopic.c_str());
120
121         if(!ret)
122         {
123             _err = "subscribe(): failed.";
124             Log::error(_err);
125         }
126
127         return ret;
128     }
129
130     bool unsubscribe(bool a=false)
131     {
132         bool ret = true;
133         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/wid/get").c_str());
134         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/wid/set").c_str());
135         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/temp/get").c_str());
136         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/temp/set").c_str());
137         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/wind/get").c_str());
138         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/wind/set").c_str());
139         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/humidity/get").c_str());
140         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/humidity/set").c_str());
141         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/forecast/get").c_str());
142         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/forecast/set").c_str());
143         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/topic/get").c_str());
144         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/topic/set").c_str());
145         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/activate").c_str());
146         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/deactivate").c_str());
147         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/save").c_str());
148         ret &= _mqttClient->unsubscribe(String(getMqttTopic() +"/load").c_str());
149         _mqttClient->unsubscribe(_weatherTopic.c_str());
150
151         if(!ret)
152         {
153             _err = "unsubscribe(): failed.";
154             Log::error(_err);
155         }
156
157         return ret;
158     }
159
160     bool resubcribe()
```

```
    {
        if(!unsubscribe())
            return false;
        return subscribe();
    }


    bool save(int const address)
    {
        config_t conf;
        conf.active = _active;
        strlcpy(conf.weatherTopic,(const char*)getWeatherTopic().c_str(),TOPIC_MAX_LENGTH);
        strlcpy(conf.mqttTopic,(const char*)getMqttTopic().c_str(),TOPIC_MAX_LENGTH);

        bool ret = true;

        // A fazer: dar um jeito de verificar se isso aqui vai dar certo
        EEPROM.begin(sizeof(wlconditions_t) + sizeof(config_t));
        EEPROM.put<config_t>(address,conf);
        EEPROM.put<wlconditions_t>(address+sizeof(conf),_wlcond);
        ret &= EEPROM.commit();
        EEPROM.end();

        if(!ret)
        {
            _err = "Could not save on EEPROM.";
            Log::error(_err);
        }
        return ret;
    }

    bool save(void)
    {
        return save(_eepromAdd);
    }

    bool load(int const address)
    {
        config_t conf;

        bool ret = true;

        // A fazer: dar um jeito de verificar se isso aqui vai dar certo
        EEPROM.begin(sizeof(wlconditions_t) + sizeof(config_t));
        EEPROM.get<config_t>(address,conf);
        EEPROM.get<wlconditions_t>(address+sizeof(conf),_wlcond);
        EEPROM.end();

        if(!ret)
        {
            _err = "Could not read from EEPROM.";
            Log::error(_err);
            return false;
        }

        _active = conf.active;
        setMqttTopic(String(conf.mqttTopic));
        setWeatherTopic(String(conf.weatherTopic));

        return ret;
    }

    bool load(void)
    {
        return load(_eepromAdd);
    }

    bool callback(const char* topic, const char* payload, unsigned int length)
    {
```

```
        String stopic = String(topic);
        char msg[length+1];
        for (int i = 0; i < length; i++) {
            msg[i] = (char)payload[i];
        }
        msg[length] = '\0';

        return this->callback(stopic, String(msg));
    }

    // Call this method inside your callback functions with raw arguments
    // returns false in case of error
    bool callback(String topic, String payload)
    {
        if(topic == _weatherTopic)
        {
            decide(payload);
        }
        else if(topic == getMqttTopic() +"/wid/get")
        {
            DynamicJsonDocument doc(JSON_OBJECT_SIZE(2));
            doc["min"] = _wlcond.wid[0];
            doc["max"] = _wlcond.wid[1];
            String data = "";
            serializeJson(doc,data);
            if(!_mqttClient->publish(payload.c_str(),data.c_str()))
            {
                _err = "Publish error! Could not publish <" + data + "> to topic <" + payload +
                    ↪  ">.";
                Log::error(_err);
                return false;
            }
        }
        else if(topic == getMqttTopic() +"/wid/set")
        {
            DynamicJsonDocument doc(JSON_OBJECT_SIZE(2) + 10);
            deserializeJson(doc, payload);
            if(doc["max"] > WID_MAX || doc["min"] < WID_MIN)
            {
                _err = "WeatherID values out of bonds.";
                Log::error(_err);
                return false;
            }
            _wlcond.wid[0] = doc["min"];
            _wlcond.wid[1] = doc["max"];
        }
        else if(topic == getMqttTopic() +"/temp/get")
        {
            DynamicJsonDocument doc(JSON_OBJECT_SIZE(2));
            doc["min"] = _wlcond.temp[0];
            doc["max"] = _wlcond.temp[1];
            String data = "";
            serializeJson(doc,data);
            if(!_mqttClient->publish(payload.c_str(),data.c_str()))
            {
                _err = "Publish error! Could not publish <" + data + "> to topic <" + payload +
                    ↪  ">.";
                Log::error(_err);
                return false;
            }
        }
        else if(topic == getMqttTopic() +"/temp/set")
        {
            DynamicJsonDocument doc(JSON_OBJECT_SIZE(2) + 10);
            deserializeJson(doc, payload);
            _wlcond.temp[0] = doc["min"];
            _wlcond.temp[1] = doc["max"];
        }
        else if(topic == getMqttTopic() +"/wind/get")
```

```
297         {
298             if(!_mqttClient->publish(payload.c_str(),String(_wlcond.wind).c_str()))
299             {
300                 _err = "Publish error! Could not publish <" + String(_wlcond.wind) + "> to
                        ↪ topic <" + payload + ">.";
301                 Log::error(_err);
302                 return false;
303             }
304         }
305         else if(topic == getMqttTopic() +"/wind/set")
306         {
307             _wlcond.wind = payload.toFloat();
308         }
309         else if(topic == getMqttTopic() +"/humidity/get")
310         {
311             if(!_mqttClient->publish(payload.c_str(),String(_wlcond.humidity).c_str()))
312             {
313                 _err = "Publish error! Could not publish <" + String(_wlcond.humidity) + "> to
                        ↪ topic <" + payload + ">.";
314                 Log::error(_err);
315                 return false;
316             }
317         }
318         else if(topic == getMqttTopic() +"/humidity/set")
319         {
320             _wlcond.humidity = payload.toInt();
321         }
322         else if(topic == getMqttTopic() +"/forecast/get")
323         {
324             if(!_mqttClient->publish(payload.c_str(),String(_wlcond.forecast).c_str()))
325             {
326                 _err = "Publish error! Could not publish <" + String(_wlcond.forecast) + "> to
                        ↪ topic <" + payload + ">.";
327                 Log::error(_err);
328                 return false;
329             }
330         }
331         else if(topic == getMqttTopic() +"/forecast/set")
332         {
333             _wlcond.forecast = payload.toInt();
334         }
335         else if(topic == getMqttTopic() +"/activate")
336         {
337             _active = true;
338             if(!_mqttClient->subscribe(_weatherTopic.c_str()))
339             {
340                 _err = "Could not subscribe to <" + _weatherTopic; + ">";
341                 Log::error(_err);
342                 return false;
343             }
344         }
345         else if(topic == getMqttTopic() +"/deactivate")
346         {
347             _active = false;
348             if(!_mqttClient->unsubscribe(_weatherTopic.c_str()))
349             {
350                 _err = "Could not unsubscribe from <" + _weatherTopic; + ">";
351                 Log::error(_err);
352                 return false;
353             }
354         }
355         else if(topic == getMqttTopic() +"/topic/get")
356         {
357             if(!_mqttClient->publish(payload.c_str(),getMqttTopic().c_str()))
358             {
359                 _err = "Publish error! Could not publish <" + getMqttTopic() + "> to topic <" +
                        ↪  payload + ">.";
360                 Log::error(_err);
361                 return false;
```

```
            }
        }
        else if(topic == getMqttTopic() +"/topic/set")
        {
            if(!unsubscribe())
                return false;

            String hold = getMqttTopic();

            setMqttTopic(payload);

            if(!subscribe())
            {
                _err = "Given topic is might unvalid.";
                Log::error(_err);
                setMqttTopic(hold);
                if(!subscribe())
                {
                    _err = "Could not resubscribe to MQTT topic. New given topic was discarded.
                        ↪ ";
                    Log::error(_err);
                }
                return false;
            }
        }
        else if(topic == getMqttTopic() +"/save")
        {
            if(!save(getEEPROMAddress()))
                return false;
        }
        else if(topic == getMqttTopic() +"/load")
        {
            if(!load(getEEPROMAddress()))
                return false;
        }

        return true;
    }

    String err()
    {
        String ret = _err;
        _err = "";
        return ret;
    }



protected:
    T * _mqttClient;
    String _err;

private:
    String _mqttTopic;
    String _weatherTopic = "weather";
    String _windowTopic = "smarthome/window";
    int _eepromAdd = 0;
    wlconditions_t _wlcond;
    bool _active = true;
};

#endif
```