UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

CAIO AUGUSTO SILVA

DESENVOLVIMENTO E VALIDAÇÃO DE MÓDULO DE COMUNICAÇÃO MQTT PARA A PLATAFORMA BIPES PARA APLICAÇÕES DE INTERNET DAS COISAS

SÃO CARLOS – SP
2020

CAIO AUGUSTO SILVA

DESENVOLVIMENTO E VALIDAÇÃO DE MÓDULO DE COMUNICAÇÃO MQTT PARA A PLATAFORMA BIPES PARA APLICAÇÕES DE INTERNET DAS COISAS

Trabalho de graduação apresentado ao Departamento de Computação da Universidade Federal de São Carlos, para obtenção do título de bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Rafael Vidal Aroca

São Carlos – SP
2020

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciência Exatas e de Tecnologia
Departamento de Computação

Folha de aprovação.

Membros da comissão examinadora que avaliou e aprovou a Defesa de Trabalho de Graduação do candidato Caio Augusto Silva, realizada em 16/12/2020:

Prof. Dr. Rafael Vidal Aroca
Universidade Federal de São Carlos

Prof. Dr. Edilson Reis Rodrigues Kato
Universidade Federal de São Carlos

Prof. Dr. Glauco Augusto de Paula Caurin
Universidade de São Paulo

AGRADECIMENTO

Meu sincero agradecimento a todos aqueles que me acompanharam em minha jornada. Aos amigos que fiz na universidade, que desde 2015 sempre estiveram presentes nos momentos de dificuldade e também de descontração, que tornaram os anos que passei em São Carlos boas lembranças para vida toda. Aos professores que encontrei pelo caminho, pelo conhecimento proferido, pelo apoio em atividades extracurriculares e, em especial ao meu orientador, pela ajuda na elaboração deste trabalho. À minha namorada, por todo companheirismo e suporte para vencer mais essa etapa. À minha família, por terem sempre me incentivado e dado condições para que eu pudesse chegar até aqui. Obrigado a todos que me ajudaram a tornar esse sonho uma realidade.

RESUMO

Uma tendência atual direciona para a adoção de métodos alternativos de programação sem código, que permitam o desenvolvimento de programas sem escrever sequer uma linha de código ou saber uma linguagem de programação. Uma abordagem é a utilização de blocos para constituir um paradigma visual de programação. Fazendo uma metáfora à peças de quebra cabeça, os comandos se encaixam para formar programas, e só conseguem ser encaixados nos lugares apropriados, evitando erros de lógica e sintaxe. Seguindo esta direção, surgiu o BIPES (*Block based Integrated Platform for Embedded Systems*), um ambiente *open source* que permite desenvolver, programar, compilar, implantar e testar aplicações em sistemas embarcados e dispositivos de Internet das Coisas. Totalmente baseado na WEB, não necessita da instalação de nenhum software adicional e suporta placas como ESP32, ESP8266 e Raspberry Pi. Devido a natureza do projeto, viu-se a necessidade de integrar um protocolo reconhecidamente indicado para IoT, o MQTT (*Message Queuing Telemetry Transport*). Portanto, foram desenvolvidos dois módulos (conjuntos de blocos) para o MQTT: um deles permite a completa configuração dos parâmetros da conexão e controle sobre as mensagens trocadas, portanto é direcionado a um usuário com maior domínio sobre a tecnologia e que busca mais flexibilidade; outro, batizado de EasyMQTT, permite a fácil prototipação e implantação de aplicações contendo o MQTT, sem configurações complicadas, com uma fácil visualização dos dados adquiridos, além de possuir uma *Application Programming Interface* (API) que possibilita sua integração em outras aplicações. Os módulos foram validados resolvendo um problema real do FITOTEC (Laboratório de Tecnologia Farmacêutica em Fitoprodutos) da UNESP – campus Assis, em que era necessário monitorar temperatura e umidade de uma chocadeira de ovos durante alguns dias. Para isso, foram criados dois programas, um para cada módulo desenvolvido, que foram testados na chocadeira durante dois período de tempo diferentes utilizando uma ESP8266. Os resultados comprovaram a estabilidade dos módulos desenvolvidos, inclusive sua capacidade de se recuperar de instabilidades na rede, conseguindo se recuperar e continuar comunicando após algumas quedas de conexão consecutivas. Além disso, o EasyMQTT também já foi utilizado em um minicurso de Internet das Coisas, e, no momento da escrita deste trabalho, o banco de dados do EasyMQTT já contava com 58 sessões diferentes, totalizando 87 tópicos e tendo processado o número crescente de 265.431 mensagens.

Palavras-chave: MQTT. Internet das Coisas. Programação baseada em blocos. Sistemas Embarcados. ESP8266.

ABSTRACT

A current trend is towards the adoption of alternative no-code programming methods, which allows the development of programs without writing even a line of code or knowing a programming language. One approach is to use blocks to constitute a visual programming paradigm. Making a metaphor for puzzle pieces, the commands snap together to form programs, and can only be joined in the appropriate places, avoiding errors of logic and syntax. Following this direction, the BIPES (Block based Integrated Platform for Embedded Systems) emerged as an open source environment that allows to develop, program, compile, deploy and test applications on embedded systems and Internet of Things devices. Totally based on the WEB, it does not require the installation of any additional software and supports boards such as ESP32, ESP8266, Raspberry Pi. Due to the nature of the project, there was a need to integrate a protocol admittedly appropriate for IoT, the MQTT (Message Queuing Telemetry Transport). Therefore, two modules (sets of blocks) were developed for the MQTT: one of them allows the complete configuration of the connection parameters and control over the messages exchanged, therefore it is aimed at a user with greater knowledge over the technology and who seeks more flexibility; another, nicknamed EasyMQTT, allows easy prototyping and deployment of applications containing MQTT, without complicated configurations, with an easy view of the acquired data, in addition to having an Application Programming Interface (API) that allows its integration in other applications. The modules were validated by solving a real problem of FITOTEC (Laboratory of Pharmaceutical Technology in Phytoproducts) located at UNESP – Assis campus, where it was necessary to monitor the temperature and humidity of an egg incubator for a few days. For this, two programs were created, one for each module developed, which were tested in the incubator for two different periods of time using an ESP8266. The results proved the stability of the developed modules, including their ability to recover from instabilities in the network, managing to recover and continue communicating after some consecutive connection failures. In addition, EasyMQTT has already been used in a short course about Internet of Things, and, at the time of writing this work, the EasyMQTT database already had 58 different sessions, totaling 87 topics and having processed an increasing number of 265,431 messages.

Keywords: MQTT. Internet of Things. Block Based Programming. Embedded Systems. ESP8266.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

API – *Application Programming Interface* ou Interface de Programação de Aplicativos

BIPES – *Block based Integrated Platform for Embedded Systems* ou Plataforma Integrada baseada em blocos para sistemas embarcados

GND – *Ground* ou Terra (eletricidade)

GPIO – *General Purpose Input/Output* ou Entrada/Saída de Propósito Geral

HTTP – *Hypertext Transfer Protocol* ou Protocolo de Transferência de Hipertexto

I2C – *Inter-Integrated Circuit* ou Circuito Interintegrado

IDE – *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado

ID – *Identification* ou Identificação

IO – *Input/Output* ou Entrada/Saída

IoT – *Internet of Things* ou Internet das Coisas

IP – *Internet Protocol* ou Protocolo de Internet

JSON – *JavaScript Object Notation* ou Notação de Objetos JavaScript

LCD – *Liquid Crystal Display* ou Tela de Cristal Líquido

MISO – *Master Input/Slave Output* ou Mestre Entrada/Escravo Saída

MOSI – *Master Output/Slave Input* ou Mestre Saída/Escravo Entrada

MQTT – *Message Queuing Telemetry Transport* ou Transporte de Filas de Mensagem de Telemetria

PHP – *Hypertext Preprocessor ou Processador Hipertexto*

PWM – *Pulse Width Modulation* ou Modulação de Largura de Pulso

QoS – *Quality of Service* ou Qualidade de Serviço

SCK – *Synchronization Clock* ou Clock de Sincronização

SPI – *Serial Peripheral Interface* ou Interface Periférica Serial

SSH – *Secure Shell*

TCP – *Transmission Control Protocol* ou Protocolo de Controle de Transmissão

TTL – *Transistor–transistor logic* ou Lógica Transistor-transistor

UART – *Universal Asynchronous Receiver/Transmitter* ou Receptor/Transmissor Universal Assíncrono

UR – Umidade Relativa

USB – *Universal Serial Bus* ou Porta Universal

VIN – *Voltage Input* ou Entrada de Tensão

Wi-Fi – *Wireless Fidelity* ou Fidelidade sem fio

XML – *Extensible Markup Language* ou Linguagem Extensível de Marcação Genérica

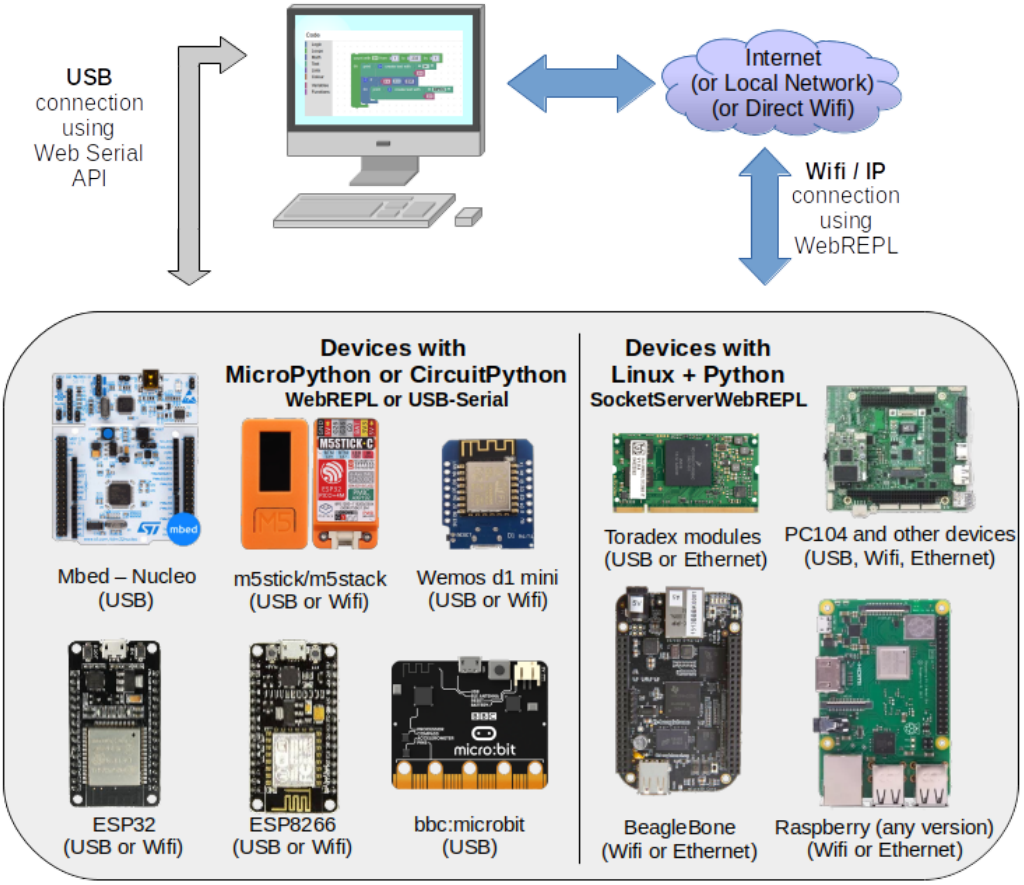
SUMÁRIO

1 INTRODUÇÃO

O BIPES (*Block based Integrated Platform for Embedded Systems*) é um projeto 100% *open source*, licenciado sob a GNU *Public Licence*, que visa tornar acessível a qualquer um desenvolver, programar, compilar, implantar e testar aplicações em sistemas embarcados e dispositivos de IoT (*Internet of Things*). Um grande diferencial do projeto é ser uma aplicação totalmente WEB. Assim, não necessita da instalação ou configuração de softwares adicionais e pode ser acessado a partir de qualquer dispositivo que disponha de um navegador WEB moderno. Toda a programação no BIPES é realizada através de blocos, o que torna o desenvolvimento de soluções mais rápido e acessível para pessoas sem muita experiência em programação (BIPES, 2020).

Os possíveis fluxos de desenvolvimento através do BIPES são representados na Figura 1. Resumidamente, o usuário monta seu programa em blocos através da plataforma, conecta o seu dispositivo via USB ou Wi-Fi e executa o programa desenvolvido.

Figura 1 – Diagrama geral de arquitetura e operação do BIPES



Fonte: JUNIOR et al., 2020

O BIPES já contava com alguns blocos que permitiam a conexão dos dispositivos à internet e a realização de requisições HTTP básicas, porém, como a ideia principal do projeto é tornar mais simples o acesso ao IoT, viu-se a necessidade de se integrar um protocolo reconhecidamente ideal para internet das coisas. O MQTT (*Message Queuing Telemetry Transport*) é um protocolo de mensagens baseado em *publish/subscribe*, extremamente simples e leve, portanto ideal para dispositivos com processamento e consumo energético limitados, redes instáveis ou com baixa largura de banda e alta latência (MQTT, 2020).

1.1 OBJETIVO

Este trabalho teve como objetivo contribuir para o desenvolvimento da plataforma *open source* BIPES. Assim, implementou blocos que possibilitem a utilização do protocolo MQTT para comunicação, tanto para enviar como para receber dados, de forma simples e fácil ao usuário. A preparação e configuração de um servidor MQTT, desenvolvimento de interface de visualização também foram contempladas. Foram desenvolvidos dois conjuntos de blocos: um deles permite a completa configuração dos parâmetros da conexão e controle sobre as mensagens trocadas, portanto é direcionado a um usuário com maior domínio sobre a tecnologia e que busca mais flexibilidade; outro, apelidado de EasyMQTT, permite a fácil prototipação e implantação de aplicações contendo o MQTT, sem configurações complicadas e com uma fácil visualização dos dados adquiridos. Também, buscou-se aprimorar alguns outros blocos, como os de leitura de sensores de temperatura DHT11/22 e de definição de pinos digitais, de forma a torná-los mais funcionais e amigáveis ao usuário. Para validação da implementação foram desenvolvidas aplicações IoT de monitoramento de umidade e temperatura empregando os módulos MQTT. Tais aplicações foram utilizadas por dois períodos em uma chocadeira de ovos, cada período utilizando um dos conjuntos de blocos desenvolvidos.

1.2 ORGANIZAÇÃO DO TRABALHO

No Capítulo 2 é realizada uma análise teórica sobre o que é programação em blocos, porque é importante e como pode ser aplicada para sistemas embarcados.

Além disso, é contextualizada a proposta do BIPES neste meio e a necessidade que motivou o desenvolvimento deste trabalho. Em seguida, é feita uma análise comparativa entre o BIPES e outros projetos similares de desenvolvimento em blocos para embarcados e IoT. Então, são introduzidos alguns conceitos chave utilizados no desenvolvimento deste trabalho.

O Capítulo 3 contém o detalhamento técnico de todo o processo de desenvolvimento dos módulos MQTT e EasyMQTT, incluindo definição dos blocos, programação da tradução para código Python, configuração de servidores, desenvolvimento de API e *Dashboard* e realização do *deploy*.

O processo de validação dos módulos desenvolvidos, bem como uma análise sobre os dados obtidos pode ser conferida no Capítulo 4.

Por fim, o código fonte contribuído ao BIPES, os códigos gerados automaticamente pelos programas de teste e uma tabela da análise de sessões em uso no EasyMQTT estão disponíveis nos Apêndices.

2 FUNDAMENTAÇÃO TEÓRICA

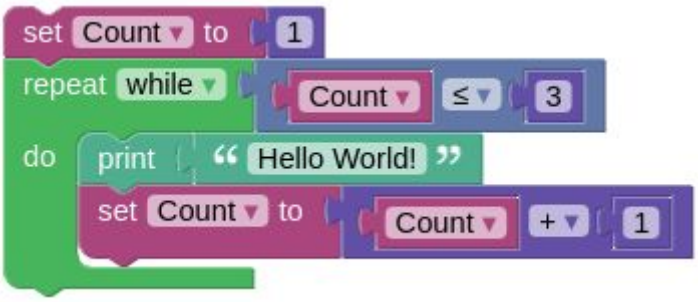
2.1 CONTEXTUALIZAÇÃO

Escrever código sempre foi a base para o desenvolvimento da maioria das aplicações que utilizamos, mas, conforme elucidado por Caballar (2020), uma tendência atual direciona para a adoção de métodos alternativos de programação sem código, que permitam o desenvolvimento sem escrever sequer uma linha de comando. A programação sem códigos permite que pessoas consigam desenvolver programas sem saber de fato uma linguagem de programação (CABALLAR, 2020).

Atualmente, já existem diversas abordagens para permitir o desenvolvimento sem código. Uma abordagem bastante utilizada é através da programação baseada em blocos, em que o usuário desenvolve a lógica do programa arrastando e juntando blocos ao invés de escrever código. Esta alternativa tem se comprovado intuitiva para muitos usuários, permitindo que até crianças que nunca tiveram contato anterior com programação possam desenvolver seus próprios aplicativos com uma curva de aprendizagem muito reduzida (JUNIOR et al., 2020).

De fato, a programação baseada em blocos é cada vez mais a forma como iniciantes estão sendo introduzidos ao universo da ciência da computação, sendo um primeiro contato à prática da programação (WEINTROP, 2019). Nos últimos anos, essa prática se difundiu com o surgimento de uma nova geração de ferramentas como o Scratch, Alice e Blockly (WEINTROP & WILENSKY, 2018). Dentre algumas características que distinguem a programação baseada em blocos da programação tradicional, baseada em escrita de código ou de outras formas visuais de programação, é a utilização da metáfora de peças de quebra cabeça como forma primitiva para programação, fornecendo ao usuário dicas visuais sobre como e onde cada comando deve ser usado. As peças, que são arrastadas para a posição desejada, se juntam para formar um programa, como ilustrado pela Figura 2. Se dois comandos não devem ser usados juntos eles não conseguirão se encaixar. Isso previne erros de sintaxe ao mesmo tempo que ainda permite o desenvolvimento da habilidade de desenvolver programas pela associação de instruções (WEINTROP, 2019).

Figura 2 – Exemplo de programação baseada em blocos



Fonte: Google Blockly[1]

Além de ser uma opção para introdução à lógica de programação, a programação sem código pode de fato permitir que pessoas criem aplicações de forma rápida ao mesmo tempo que apresentem funcionalidades reais da mesma forma que uma aplicação desenvolvida por um engenheiro de software apresentaria (CABALLAR, 2020).

Em nosso dia-a-dia estamos rodeados por dispositivos eletrônicos. Impressoras, câmeras, televisores, robôs, carros, dentre outras centenas de produtos, todos possuem em comum uma característica: são sistemas embarcados e são controlados por um tipo especial de software chamado de software embarcado (JUNIOR et al., 2020). Portanto, um sistema embarcado pode ser definido como um sistema microprocessado (assim como um computador pessoal) utilizado para funções específicas (POZZEBOM, 2014).

Pode-se dizer que o surgimento de dispositivos como o Arduino provocaram uma revolução, pois devido à seu baixo custo e uma interface de programação mais simplificada, dentre vários fatores, tornou mais fácil o acesso à dispositivos embarcados para aprendizagem e prototipação. É difícil mensurar todas as possibilidades de desenvolvimento com tais dispositivos. Alguns exemplos são: Automatizar uma casa, carro, escritório; Automatizar os processos de manufatura da indústria; Criar novos brinquedos ou melhorar os já existentes; Ou até mesmo criar dispositivos inteligentes conectados a rede de computadores, conceito conhecido como IoT (MADEIRA, 2018).

Apesar do interesse crescente na área de IoT, robótica e sistemas embarcados em todos os níveis de educação, ainda nota-se a falta de ferramentas mais simples e intuitivas para programação de tais dispositivos, tanto para educação

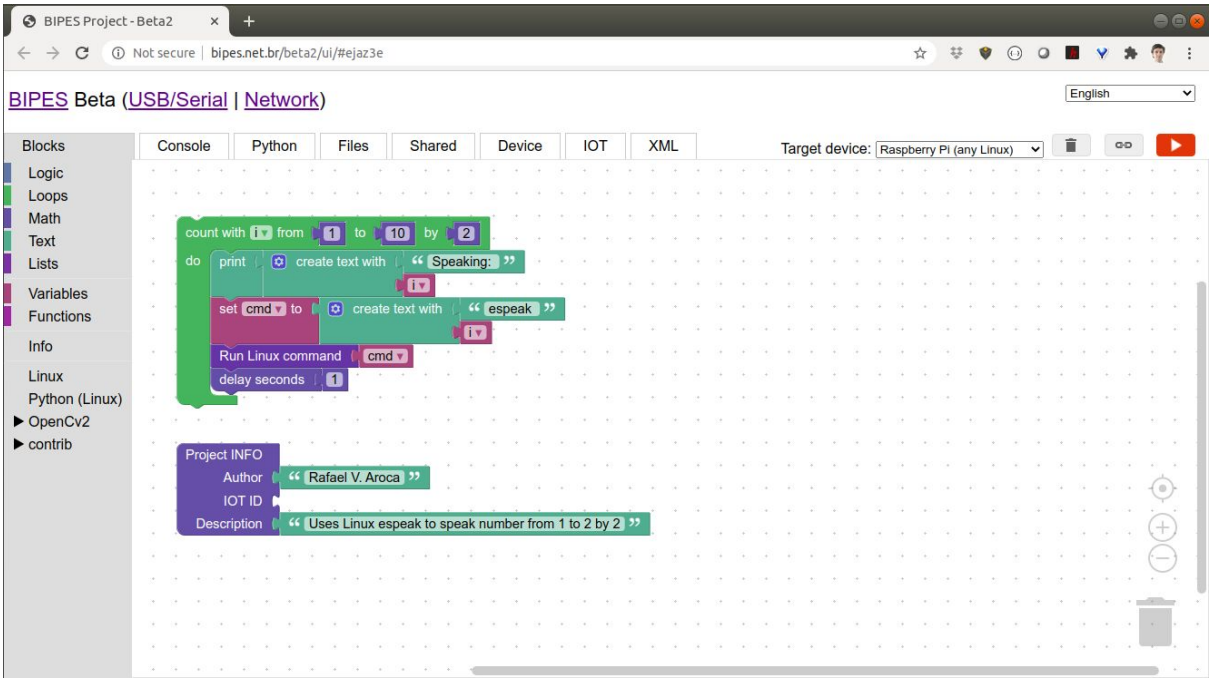
[1] Disponível em: <https://developers.google.com/blockly>. Acesso em: 26 nov. 2020.

quanto para uso profissional. Para se programar a maioria das placas normalmente utilizadas para essas aplicações é necessário baixar e instalar softwares de desenvolvimento, muitas vezes de uma forma pouco trivial ao usuário menos experiente. Ainda, as placas que são direcionadas a permitir uma utilização mais fácil costumam ser mais caras e difíceis de se encontrar (JUNIOR et al., 2020).

Seguindo nessa direção, surgiu o BIPES (*Block based Integrated Platform for Embedded Systems*), que consiste na proposta de oferecer um plataforma de desenvolvimento que suporta diversas placas e é totalmente baseada na WEB, portanto não requer a instalação de qualquer *plug-in* ou software adicional. O BIPES gera código Python que, através do MicroPython, CircuitPython ou Python, pode ser executado em placas como ESP32, ESP8266, Raspberry Pi, dentre outras (JUNIOR et al., 2020).

O BIPES já contava com diversos blocos para programação, tanto de lógica básica, quanto funções mais avançadas, como comunicação I2C, SPI, OneWire e UART Serial (JUNIOR et al., 2020). A interface do BIPES está demonstrada na Figura 3.

Figura 3 – Interface do BIPES

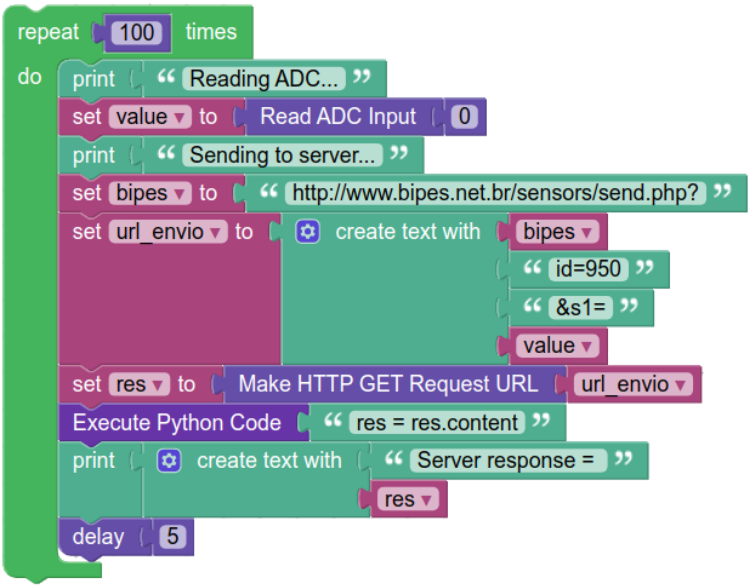


Fonte: JUNIOR et al., 2020

O BIPES também já possuía uma interface simples para IoT, em que é possível enviar dados para o servidor através de requisições HTTP do tipo GET e

visualizá-las graficamente usando a guia “IOT” presente na interface. O envio dos dados a partir do placa a ser programada podia ser feito utilizando os blocos de Requisição HTTP, conforme ilustrado pelo fragmento de código na Figura 4 (JUNIOR et al., 2020).

Figura 4 – Exemplo de IoT desenvolvido com o BIPES



Fonte: JUNIOR et al., 2020

O desenvolvimento do BIPES foi iniciado em Março de 2020, e, apesar de já funcional, ainda está em versão BETA, existindo diversos pontos a serem melhorados e funcionalidades a serem implementadas (JUNIOR et al., 2020). Um dos pontos sugeridos era a inclusão de suporte a um reconhecido protocolo de comunicação para IoT, o MQTT (*Message Queuing Telemetry Transport*). Tal protocolo apresenta a vantagem de ser extremamente leve e simples, projetado para dispositivos com processamento e consumo energético limitados, redes instáveis ou com baixa largura de banda e alta latência, portanto ideal para aplicações de IoT (MQTT, 2020). Desta forma, o presente projeto contribui para a iniciativa *open source* BIPES contribuindo com o desenvolvimento e validação de blocos para comunicação MQTT.

2.2 TRABALHOS RELACIONADOS

Existem diversos projetos com uma proposta similar ao BIPES: oferecer uma forma fácil de programação em blocos para dispositivos embarcados. MicroBlocks, BlockyDuino, BloPy, TUNIOT for ESP8266, Microsoft MakeCode e M5Stack UIFlow

são alguns exemplos. Entretanto, nenhum deles engloba todas as funcionalidades e contribuições que o BIPES oferece.

O Microblocks[2] oferece um firmware personalizado para diversas placas, permitindo que sejam facilmente programadas. É uma boa alternativa para aprendizado, pois conta com diversos blocos para trabalhar com display LCD, servo motores, dentre outros. Porém, apesar do suporte ao ESP8266, seu desenvolvimento é mais focado em algumas placas mais caras e difíceis de se encontrar, até mesmo pois é um ambiente direcionado para crianças e essas placas seriam mais amigáveis de se utilizar. Além disso, é necessário baixar e instalar software no computador para conseguir utilizar o Microblocks.

O BlockyDuino[3] oferece programação em blocos para o Arduino. Além dos blocos de lógica básica, possui blocos customizados para vários shields e dispositivos do Groove kit. Porém, gera código em C que deve ser copiado e compilado em um ambiente de desenvolvimento para o Arduino previamente configurado.

O Blopy[4] oferece um ambiente de desenvolvimento baseado em blocos capaz de se conectar via WEB à um interpretador Python webREPL, da mesma forma que o BIPES. Porém, possui somente blocos de lógica básica em sua interface e não apresenta nenhuma customização de forma a ajudar o usuário ou expandir as possibilidades de desenvolvimento com devices agregados e funcionalidades avançadas.

O TUNIOT for ESP8266[5] é uma IDE online que se assemelha com a proposta desse projeto. O TUNIOT conta com blocos de lógica básica, display LCD e vários específicos para IoT, inclusive suporte à MQTT. A IDE gera código na linguagem C, porém não compila nem envia o código para a placa. Desse modo, é necessário instalar um kit de desenvolvimento para a placa no computador local para conseguir compilar, gravar a placa e testar a aplicação.

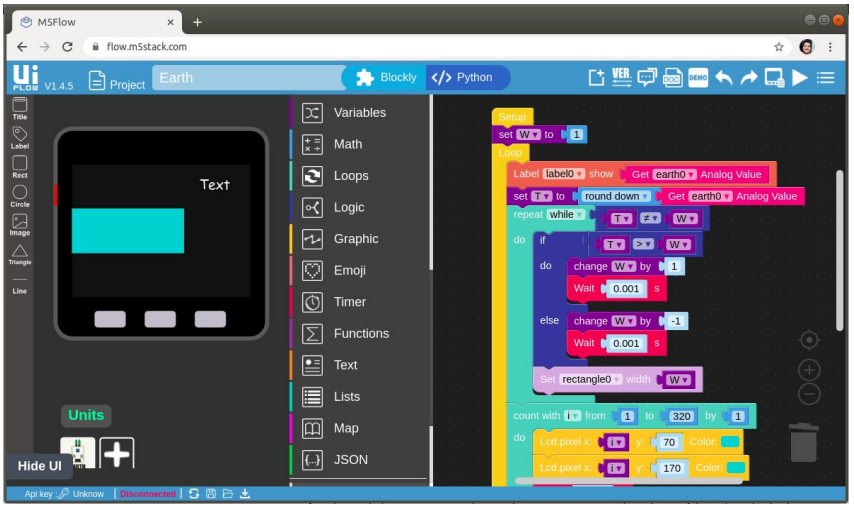
O Microsoft MakeCode[6] é um ambiente *open source* de desenvolvimento em blocos que suporta algumas plataformas como o Lego Mindstorms EV3, BBC micro:bit e Adafruit Circuit Playground Express. Não necessita da instalação de

[2] Disponível em: <https://microblocks.fun/>. Acesso em: 26 nov. 2020.
[3] Disponível em: <https://github.com/BlocklyDuino/BlocklyDuino>. Acesso em: 26 nov. 2020.
[4] Disponível em: <https://github.com/mnoriaki/Blopy>. Acesso em: 26 nov. 2020.
[5] Disponível em: <http://easycoding.tn/esp32/demos/code/>. Acesso em: 26 nov. 2020.
[6] Disponível em: <https://www.microsoft.com/pt-br/makecode/>. Acesso em 26 nov. 2020.

nenhum software e funciona direto do navegador. É muito amigável, sendo bastante alinhado com a proposta do BIPES, porém se limita a algumas placas de desenvolvimento mais caras.

A M5Stack[7] desenvolve módulos empilháveis baseados no ESP32 que podem ser utilizados para diferentes aplicações, incluindo IoT. Seus dispositivos utilizam um firmware próprio, baseado em MicroPython, que executa aplicações em Python. O UIFlow é a IDE de desenvolvimento em blocos desenvolvida pela M5Stack possui uma interface bastante atrativa e gera código Python compatível com seus produtos. Para executar o código nos módulos é necessário primeiro conectá-los à internet e atrelar uma chave da API entre produto e IDE. O UIFlow apresenta blocos para diversas funcionalidades, inclusive o MQTT. Essa é uma proposta muito parecida com o BIPES, entretanto para utilizá-la é necessário ter uma placa fabricada pela M5Stack com o firmware próprio, o que é um grande fator limitante não apresentado pelo BIPES. A Figura 5 apresenta uma tela do UIFlow.

Figura 5 – Interface do M5Stack UIFlow



Fonte: Elaborado pelo autor.

O maior diferencial do BIPES é ser uma plataforma que oferece tudo que é necessário para desenvolver, rodar e testar aplicações ricas em funcionalidades em um só lugar, sem que o usuário precise baixar ou instalar nada, sendo compatível com dispositivos acessíveis e baratos, e isso nenhum dos trabalhos relacionados apresenta.

[7] Disponível em: <https://m5stack.com/>. Acesso em 26 nov. 2020.

2.3 REFERENCIAL TÉCNICO

O BIPES é uma aplicação web escrita primariamente em Javascript e que gera código Python. O projeto utiliza-se de código proveniente de diversos outros projetos open-source como base, como o Google Blockly e MicroPython (JUNIOR et al., 2020). O BIPES é capaz de programar uma variedade de placas, como o ESP8266, ESP32, BeagleBone, mBed e RaspberryPi. Porém, para o contexto deste trabalho, a validação do desenvolvimento se limita apenas ao ESP8266.

2.3.1 Google Blockly

O Google Blockly é uma biblioteca JavaScript para construção de editores de programação baseada em blocos, capaz de gerar código semanticamente correto na linguagem escolhida. Foi introduzida em Maio de 2012 pelo Google e, desde então, segue em constante desenvolvimento pelo mesmo. O Blockly permite a criação de blocos e *toolboxes* customizadas, que se conectam de acordo com padrões escolhidos pelo desenvolvedor (JUNIOR et al., 2020). Diversos projetos utilizam o Blockly, por exemplo todos os já citados anteriormente exceto o MicroBlocks.

No contexto do BIPES, o Blockly é utilizado para o ambiente de desenvolvimento, permitindo a criação dos programas e gerando código Python automaticamente (JUNIOR et al., 2020).

Para auxiliar no desenvolvimento deste projeto, a documentação oficial do Blockly[8] foi bastante utilizada.

2.3.2 MicroPython

O Python é uma linguagem de alto nível, *open source* e de fácil aprendizado utilizada em milhares de aplicações reais ao redor do mundo, incluindo sistemas grande porte e de missão crítica (Python.org, 2020).

O MicroPython é uma implementação do Python 3 reduzida e otimizada para funcionar em microcontroladores e ambientes restritos. O MicroPython contém um subconjunto das funções padrões do Python, além de diversas funções adicionais

[8] Disponível em: <https://developers.google.com/blockly/guides/overview>. Acesso em 26 nov. 2020.

avançadas, ao mesmo tempo que necessita de somente 256 KB de espaço de armazenamento e 16 KB de RAM para ser executado.

Contendo compilador e tempo de execução completos, o Micropython é capaz de rodar comandos instantaneamente através do console interativo via conexão serial (USB) tanto quanto executar scripts armazenados no sistema de arquivos integrado. O MicroPython tenta ser o mais compatível possível com Python comum, a fim de permitir que um código que funciona no ambiente desktop possa ser facilmente transferido para um sistema embarcado (MicroPython, 2020).

O MicroPython é o firmware base necessário para utilização do BIPES na ESP8266, ESP32 e placas similares (BIPES, 2020). Portanto, para o contexto deste projeto, o ESP8266 funciona com MicroPython .

2.3.3 *Internet of Things* e MQTT

O IoT, do inglês *Internet of Things*, é um conceito que se refere a objetos e máquinas que utilizam a Internet para se comunicarem entre si, ao contrário da forma humano-humano de comunicação usual (Tan e Wang, 2010). Em 2010, Tan e Wang já diziam sobre as “coisas” que conseguem trocar informações entre si e que o número de “coisas” conectadas à Internet em um futuro próximo seria maior do que o número de pessoas.

Os dispositivos inteligentes, como são chamados esses objetos conectados à Internet, já estão por todo lugar. Relógios, geladeiras, carros e casas automatizadas, dentre inúmeros outros exemplos, esses dispositivos têm a capacidade de trocar dados na internet, seja provendo sensoriamento ou tomando ações automaticamente, por exemplo.

Dada a aplicação, é necessária uma infraestrutura que permita a sua implantação. Assim, para a comunicação de dispositivos IoT, um protocolo bastante utilizado é o MQTT (*Message Queuing Telemetry Transport*). O protocolo MQTT oferece uma arquitetura de comunicação diferente da abordagem padrão de requisição-resposta através da utilização de um modelo publicador-subscritor assíncrono. O MQTT é um protocolo padronizado pela Comitê Técnico OASIS reconhecido para utilização em IoT. É de implementação muito simples, oferece QoS (*Quality of Services*) para a comunicação com a rede e garante confiabilidade e

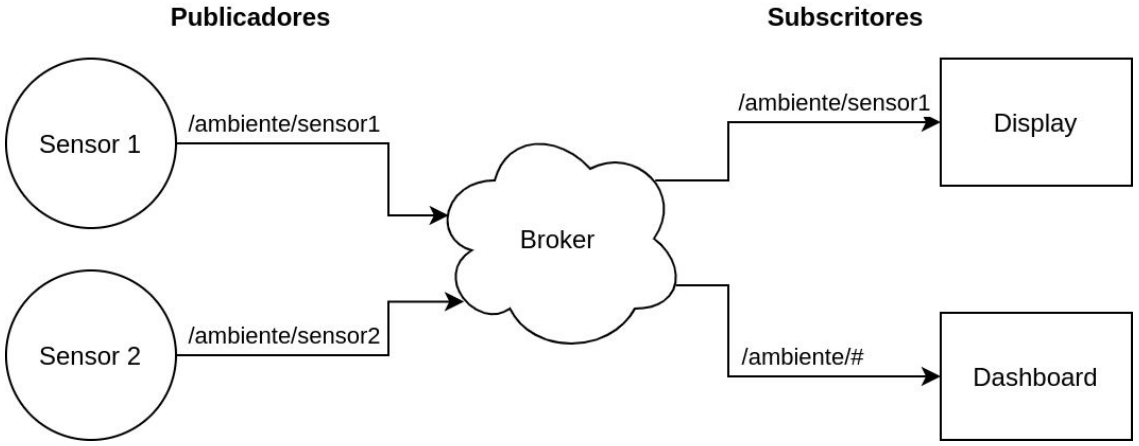
algum grau de garantia de entrega das mensagens. Por padrão, o MQTT utiliza a porta 1883 TCP/IP (MQTT, 2020).

No modelo cliente-servidor tradicional, os clientes se comunicam diretamente ao seu ponto de destino, enviando uma requisição e aguardando uma resposta. Já o modelo publicador-subscritor funciona de maneira diferente, essa arquitetura desacopla o cliente que enviou a mensagem (publicador) dos clientes que estão interessados em recebê-la (subscritores). Um publicador nunca entra em contato diretamente com um subscritor, mas sim com um "*broker*", que nada mais é do que um servidor responsável receber as mensagens, filtrá-las e distribuí-las corretamente aos subscritores (HiveMQ, 2015).

O MQTT utiliza um sistema de organização de mensagens baseado em tópicos. Sendo assim, cada mensagem deve estar identificada por um tópico para que o "*broker*" consiga direcioná-la aos subscritores certos. A mensagem MQTT contém um *payload* binário, desse modo, fica a cargo da implementação escolher qual formato de dado mais convém ser utilizado, podendo ser um JSON, XML, String, ou outros que o programador desejar. Os nomes dos tópicos são estruturados hierarquicamente utilizando o caractere barra "/" para delimitação, como uma analogia a um sistema de arquivo usual, e o caractere curinga é o "#" (HiveMQ, 2015).

A Figura 6 exemplifica um modelo que utiliza o MQTT. Os sensores são publicadores que escrevem cada um em um tópico diferente dentro do tópico ambiente. Um dos subscritores, o "Display", está interessado em receber as atualizações somente do "Sensor 1", portanto, subscreve-se somente ao tópico "/ambiente/sensor1", já o "Dashboard" gostaria de receber qualquer mensagem enviada ao tópico "ambiente", assim subscreve-se ao tópico "/ambiente/#".

Figura 6 – Exemplo de um modelo que utiliza o MQTT



Fonte: Elaborado pelo autor.

Adicionalmente, cada mensagem no MQTT conta com a definição do nível de *Quality of Service* (QoS) que se espera do serviço. O nível de QoS é um acordo entre o transmissor e o receptor da mensagem que define o nível de garantia necessário. Sendo uma característica chave do MQTT, o QoS permite ao cliente (publicador e subscritor) o poder de escolher um nível de garantia que melhor se encaixe à sua realidade de confiabilidade de conexão e lógica de aplicação. Dessa forma, o MQTT consegue tornar a comunicação em ambientes de rede não confiável muito mais fácil, ficando encarregado da retransmissão de mensagens quando necessário e garantindo a entrega (HiveMQ, 2015). Os possíveis valores de QoS aplicados a uma mensagem estão demonstrados na Tabela 1.

Tabela 1 – Níveis de QoS do MQTT

Nível de QoS	Significado
0	No máximo uma vez
1	Pelo menos uma vez
2	Exatamente uma vez

Fonte: HiveMQ, 2015.

Um nível de QoS 0 significa que não há garantia da entrega da mensagem, ela será enviada somente uma vez, não será solicitada uma confirmação de recebimento e não será armazenada. O nível 1 garante que a mensagem será entregue pelo menos uma vez, pois solicita uma confirmação de recebimento, portanto a mensagem é armazenada até que todos os interessados em recebê-la confirmem o recebimento. Um problema desse nível é que a mesma mensagem

pode ser recebida mais de uma vez caso somente a confirmação de recebimento tenha deixada de ser entregue. Já o nível de QoS 2 garante que cada mensagem seja entregue somente uma vez utilizando um *handshake* de confirmação de recebimento de quatro etapas, sendo mais lento que as outras opções apesar de mais confiável (HiveMQ, 2015).

Os módulos MQTT para o BIPES desenvolvidos nesse trabalho utilizaram como base a biblioteca umqtt.robust[9], que é um cliente MQTT desenvolvido especificamente para o MicroPython. O umqtt suporta os níveis 0 e 1 de QoS. O nível 2 ficou de fora da implementação devido à limitação de memória a que a biblioteca está submetida. A versão "robust" do umqtt tenta tornar a implementação um pouco mais robusta e confiável através da definição de algumas funções de auto-reconexão e retransmissão em caso de falhas na rede. Os métodos implementados pelo umqtt estão detalhadas na Tabela 2.

Tabela 2 – Métodos implementados pelo umqtt

Método	Descrição
connect(clean_session= True)	Conecta ao servidor.
disconnect()	Desconecta do servidor, liberando recursos.
ping()	Faz um ping ao *broker*.
publish(topic, msg, retain=False, qos=0)	Publica uma mensagem.
subscribe(topic)	Subscreve a um tópico
set_callback(f)	Define o *callback* para o recebimento de mensagens subscritas.
set_last_will(topic, msg, retain=False, qos=0)	Define a mensagem "*last will*" do MQTT.
wait_msg()	Espera por uma mensagem do broker (bloqueante). Mensagens subscritas serão entregues ao *callback*.
check_msg()	Verifica se existe alguma mensagem pendente do broker (não bloqueante). Mensagens subscritas serão entregues ao *callback*.

Fonte: Página do umqtt.simple no GitHub[10].

[9] Disponível em: <https://github.com/micropython/micropython-lib/tree/master/umqtt.robust>. Acesso em: 26 nov. 2020.

2.3.4 Armazenamento de dados e exibição

Como citado anteriormente, no contexto desse projeto, os dados serão enviados a partir do sistema embarcado utilizando o protocolo MQTT. Porém, esse protocolo tem a função somente de transmitir a informação, portanto, a fim de armazená-la e torná-la útil se faz necessário o emprego de alguma solução.

A primeira abordagem que foi utilizada neste projeto foi a utilização da plataforma Thingspeak desenvolvida pela MathWorks. Essa plataforma permite agregar, visualizar e analisar fluxos de dados ao vivo na nuvem, além de permitir a integração com serviços WEB como o Twitter e fazer análises integradas ao MATLAB. O Thingspeak[11] permite a prototipação e construção de sistemas de IoT sem a necessidade de configuração de servidores ou desenvolvimento WEB (MathWorks, 2020). Portanto, essa plataforma foi escolhida por permitir uma fácil validação inicial do funcionamento dos blocos MQTT desenvolvidos, além de oferecer o armazenamento e visualização dos dados na nuvem.

A segunda abordagem partiu do pressuposto de configurar um sistema independente para os blocos EasyMQTT, que permitissem a prototipação de aplicações IoT sem nem ao menos precisar se preocupar com a criação de conta em algum serviço como o Thingspeak ou com a configuração dos parâmetros para o funcionamento do MQTT, sendo uma forma de oferecer um primeiro contato agradável com essa tecnologia para um usuário pouco experiente. Para isso, foi utilizado o servidor na nuvem do BIPES. Nele, foi efetuada a configuração do *broker* MQTT Mosquitto, um banco de dados MongoDB, desenvolvidos scripts em Python para subscrição e armazenamento dos dados recebidos via MQTT no banco de dados e uma API em PHP para disponibilizar a visualização dos dados integrada à plataforma do BIPES.

[10] Disponível em: <https://github.com/micropython/micropython-lib/tree/master/umqtt.simple>. Acesso em: 26 nov. 2020.
[11] Disponível em: <https://thingspeak.com>. Acesso em 21 dez. 2020.

3 MATERIAIS E MÉTODOS

No desenvolvimento deste projeto foram utilizados uma placa contendo o módulo ESP8266, um módulo sensor de temperatura DHT11 e um módulo Relé para a aplicação de validação.

Foram desenvolvidos dois conjuntos de blocos de comunicação MQTT para o BIPES, chamados de Módulo MQTT e Módulo EasyMQTT. Cada um deles apresenta uma proposta diferente e é direcionado a usuários com nível de conhecimento diferentes, sendo o EasyMQTT uma solução para um primeiro contato simples e descomplicado com IoT.

3.1 MÓDULO ESP8266

O módulo ESP8266 é uma opção muito utilizada quando se diz a respeito de IoT devido ao seu baixo custo, baixo consumo de energia, WiFi e stack TCP/IP integrada, facilidade de programação e alimentação por USB (Espressif Systems, 2020). Nesse projeto foi utilizado o módulo ESP8266 NodeMCU V3 (Figura 7), que é uma placa criada para facilitar o desenvolvimento de aplicações com o ESP8266 por já conter todos os circuitos necessários para o chip funcionar, incluindo a interface Serial-USB usada para programação e depuração, regulador de tensão e barramentos de pinos compatível com Protoboard (Eletrogate, 2020), além de ser muito fácilmente encontrada no Brasil a um preço baixo. As especificações detalhadas dessa placa estão disponíveis na Tabela 3 e a sua pinagem na Figura 8.

Figura 7 – Placa ESP8266 NodeMCU V3

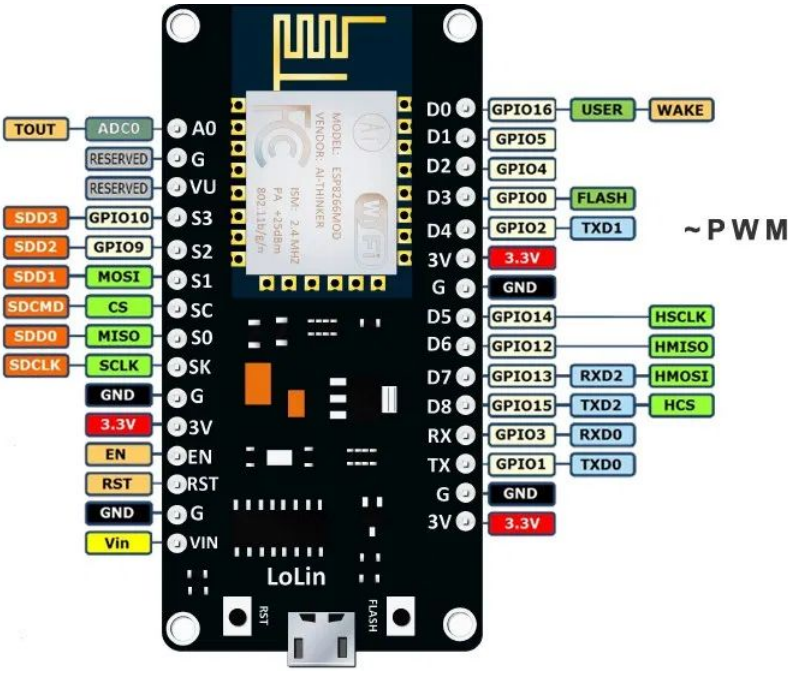
Fonte: Eletrogate, 2020

Tabela 3 – Especificações do ESP8266 NodeMCU

Parâmetro	Valor
Microcontrolador	ESP8266-12E
CPU	Tensilica Xtensa LX106 RISC 32 bits
Clock do Processador	80 MHz – 160 MHz
Memória RAM	20 KB
Memória Flash	4 MB
Wi-Fi	2.4 Ghz 802.11 b/g/n 300Mbps com antena embutida
Pinos ADC	1 (10 bits de resolução)
Pinos GPIO	10 (incluindo MISO, MOSI, SCK, PWM, I2C, SPI e RX, TX)
Tensão de alimentação	5V à 9V (Via MicroUSB e pino VIN)
Corrente de operação	70 mA em média (200 mA pico)
Temperatura de operação	-40ºC à +125ºC

Fonte: Eletrogate, 2020 e Espressif Systems, 2020

Figura 8 – Pinagem da placa ESP8266 NodeMCU V3



Fonte: Microcontroller Tutorials[12]

[12] Disponível em: <https://www.teachmemicro.com/nodemcu-pinout/>. Acesso em: 21 de novembro de 2020.

É possível adquirir placas já pré-carregadas com o MicroPython em sites como o Adafruit[13]. Ademais, é uma funcionalidade planejada do BIPES o desenvolvimento de uma interface que permita a instalação do MicroPython nas placas de forma fácil e automática através do navegador, porém ainda não está implementada. Assim, para que a ESP8266 NodeMCU utilizada no presente projeto pudesse ser programada com o BIPES, a placa precisou ser inicialmente carregada com o MicroPython manualmente.

3.1.1 Instalação do MicroPython

Para a instalação do MicroPython na ESP8266, primeiramente é necessário instalar o “esptool”, uma ferramenta baseada em Python utilizada para se comunicar com o bootloader presente nas placas ESP da Espressif Systems. Para tal, foi executado o seguinte comando no terminal Linux:

```
sudo pip install esptool
```

Após a instalação, foi necessário obter o binário do MicroPython a partir de sua página de downloads[14]. Foi escolhida a última versão estável disponível que, no momento da escrita, corresponde à “esp8266-20200911-v1.13.bin”.

A placa foi então conectada ao computador através da interface USB e foram executados os seguintes comandos para apagar a memória flash e carregar o firmware contendo o MicroPython.

```
esptool.py --port /dev/ttyUSB0 erase_flash
esptool.py --port /dev/ttyUSB0 --baud 460800 write_flash
--flash_size=detect 0 esp8266-20200911-v1.13.bin
```

É interessante notar que nesse caso a porta serial associada a placa foi /dev/ttyUSB0, mas pode ser diferente em outras ocasiões.

Sendo o processo bem sucedido, a placa está pronta para uso.

3.2 MÓDULO SENSOR DHT11

O sensor de temperatura e umidade DHT11 (Figura 9) foi utilizado na

[13] Disponível em: <https://www.adafruit.com/>. Acesso em: 20 nov. 2020.
[14] Disponível em: <http://micropython.org/download/esp8266/>. Acesso em 20 nov. 2020.

aplicação de validação das implementações dos módulos MQTT. Suas especificações técnicas estão disponíveis na Tabela 4.

Figura 9 – Sensor de Temperatura DHT11

Fonte: Elaborado pelo autor.

Tabela 4 – Especificações do sensor de temperatura DHT11

Parâmetro	**Valor**
Tensão de Alimentação	3 V – 5 V (5,5 V Max)
Corrente de Operação	200uA – 500mA
Corrente de Stand-by	100uA – 150uA
Faixa de Medição de Umidade	20 – 90%UR
Precisão da medição de Umidade	± 5% UR
Faixa de medição de Temperatura	0 – 50ºC
Precisão da Medição de Temperatura	± 2 ºC
Tempo de Resposta	< 5s
Período de Amostragem	> 2s
Dimensões	23 x 12 x 5mm (incluindo terminais)

Fonte: Baú da Eletrônica[15]

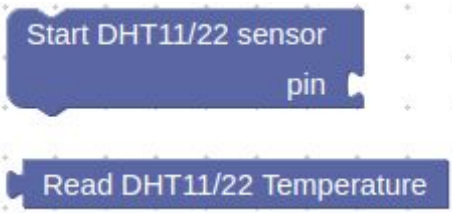
3.2.1 Blocos para sensor de temperatura DHT11/22

No início do desenvolvimento deste projeto já existiam blocos para utilização do sensor DHT11 implementados no BIPES, vide Figura 10. Porém, sua utilização estava limitada a leitura da temperatura e, apesar de informar também ser

[15] Disponível em: <https://www.baudaeletronica.com.br/sensor-de-umidade-e-temperatura-dht11.html>. Acesso em: 10 nov. 2020.

compatível com o DHT22, era compatível somente com o DHT11.

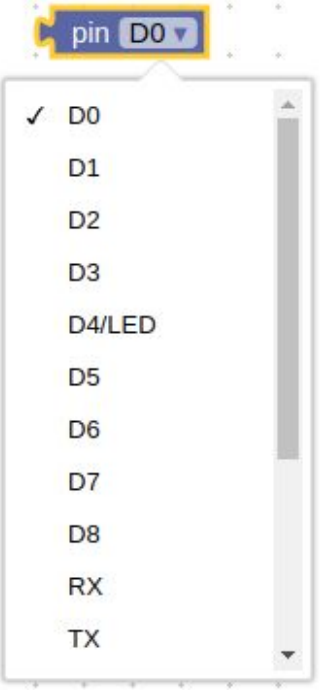
Figura 10 – Blocos anteriores do BIPES para o sensor DHT11/22



Fonte: Elaborado pelo autor.

Assim, visando aprimorar esse conjunto de blocos, foram criados e modificados alguns blocos. A fim de tornar o ambiente mais amigável ao usuário, foi primeiramente criado um bloco que permite a escolha do pino utilizado para operações de IO através de uma lista, ao invés de exigir que usuário precisasse manualmente olhar em uma lista o número correspondente à porta utilizada. Tal bloco, representado na Figura 11, pode ser utilizado com todos os blocos do BIPES que necessitem da definição de um pino.

Figura 11 – Bloco de seleção de pino



Fonte: Elaborado pelo autor.

O funcionamento desse bloco de escolha do pino é bastante simples. Foi definido um *array* junto ao código base do aplicativo no arquivo “index.html” contendo as correspondências entre nome do pino e o número do GPIO para cada placa. No contexto deste projeto, tal função foi implementada somente para o

ESP8266, porém pode ser facilmente expandida para as outras placas através da adição de novas entradas no array demonstrado abaixo.

```
var pinout = {
  'ESP8266': [
    ['D0', "16"],
    ['D1', "5"],
    ['D2', "4"],
    ['D3', "0"],
    ['D4/LED', "2"],
    ['D5', "14"],
    ['D6', "12"],
    ['D7', "13"],
    ['D8', "15"],
    ['RX', "3"],
    ['TX', "1"],
    ['SD3', "10"],
    ['SD2', "9"],
    ['SD1', "8"],
    ['CMD', "11"],
    ['SD0', "7"],
    ['CLK', "6"]
  ]
};
```

O procedimento para adição de um novo bloco na interface é bastante simples, contando com poucas etapas. O primeiro passo foi incluir a definição do bloco no arquivo “block_definitions.js”, que é responsável por definir como cada bloco deve parecer, configurando mensagens de ajuda, entradas, saídas e demais características.

Pode-se ver abaixo que o bloco “pinout” foi configurado para ser um bloco que retorna um valor através da linha “`this.setOutput(true, null)`” e que sua única entrada é um seletor *dropdown.* O método “`update_list()`” é chamado sempre no começo da inicialização deste bloco, carregando na variável “options” o array de pinos apropriado para a placa sendo utilizada no momento, dado este obtido através da chamada “`document.getElementById('device_selector').value`”.

```
Blockly.Blocks['pinout'] = {
 update_list: function() {
   if (document.getElementById('device_selector').value in pinout){
     this.options = pinout[document.getElementById('device_selector').value];
   }else{
     this.options = [["Pins are not defined","None"]];
   }
 },
 options: [],
```

```
 init: function() {
   this.update_list();
   this.appendDummyInput()
       .appendField('pin')
       .appendField(new Blockly.FieldDropdown(this.options), 'PIN');
   this.setOutput(true, null);
   this.setColour(230);
   this.setTooltip("Pins");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};
```

Em seguida, para que os blocos pudessem ser traduzidos para código em Python foi necessário defini-lo também no arquivo “generator_stubs.js”, como pode ser visto em seguida. Este é um bloco muito simples, portanto, o valor retornado por ele será somente o valor que estava armazenado no array, representado na variável “pin”.

```
Blockly.Python['pinout'] = function(block) {
 var pin = block.getFieldValue('PIN');
  return [pin, Blockly.Python.ORDER_NONE];
};
```

Por fim, para que o bloco aparecesse na toolbox do BIPES ainda foi necessário atualizar as entradas desses blocos no XML que a define, dentro do arquivo “index.html”. Para isso, a seguinte entrada foi adicionada.

```
   <block type="pinout"></block>
```

Os blocos para o sensor DHT11/22 utilizam a biblioteca padrão do MicroPython “dht”. Nela, é necessário primeiro efetuar a medida do sensor, que obtém as leituras do sensor, para que esses valores de temperatura e umidade possam depois ser acessados. Assim, o bloco “Read DHT11/22 Temperature” inicialmente gerava o seguinte código:

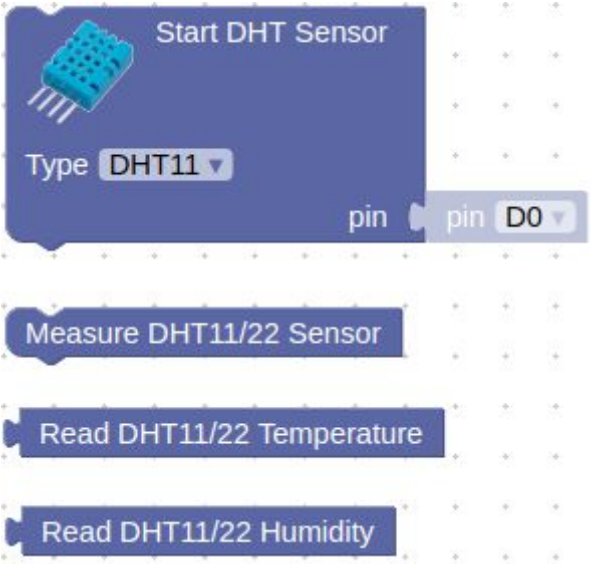
```
import dht

dhts.measure()
dhts.temperature()
```

Entretanto, atrelar o comando “`dhts.measure()`” a cada leitura de temperatura ou umidade não é uma boa ideia, pois, como detalhado nas especificações, este sensor precisa de ao menos 2 segundos entre uma medida e outra para produzir dados válidos e a biblioteca já implementou ele separadamente

justamente por isso. Se fosse deixado dessa forma, seria necessário incluir um delay dentro do bloco para permitir que o comando de leitura de temperatura e umidade pudessem ser usados em seguida, o que causaria possíveis atrasos indesejados na execução do código. Portanto, o bloco “Measure DHT11/22” Sensor foi criado.

Além disso, foi adicionado o bloco faltante para leitura de umidade “Read DHT11/22 Humidity” e aprimorado o bloco de inicialização “Start DHT Sensor” para permitir a escolha da variante de sensor a ser utilizada. O conjunto de blocos final após as modificações pode ser visto na Figura 12.

Figura 12 – Novos blocos para o sensor DHT11/22



Fonte: Elaborado pelo autor.

Da mesma forma que foi feito feito com o bloco de seleção de pinos, para definir os blocos do sensor DHT, foi necessário alterar e incluir os novos blocos no arquivo “block_definitions.js”. O bloco de inicialização cujo código está representado abaixo possui algumas definições a mais do que o exemplo anterior. Para definir que é necessário conectar um bloco determinando o pino a que o sensor está ligado foi utilizado o comando “`this.appendValueInput("pin")`” e realizada a checagem de que o valor recebido é realmente um número usando “`setCheck("Number")`”. Para permitir que o bloco pudesse ser empilhado tanto antes quanto depois de outros blocos foram feitas as definições “`this.setPreviousStatement(true, null)`” e “`this.setNextStatement(true, null)`”.

```
/// Start DHT Sensor
```

```
Blockly.Blocks['dht_init'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldImage(
                   "/beta2/ui/media/dht.png",
                   55,
                   55,
                   "*"))
       .appendField("Start DHT Sensor");
   this.appendDummyInput()
       .appendField('Type')
       .appendField(new Blockly.FieldDropdown([
                   ['DHT11', 'DHT11'],
                   ['DHT22', 'DHT22']
       ]), 'DHT_TYPE');
   this.appendValueInput("pin")
       .setCheck("Number")
       .setAlign(Blockly.ALIGN_RIGHT)
       .appendField(new Blockly.FieldLabelSerializable("pin"), "DHT_PIN_MSG");
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setTooltip("Start DHT11 ou DHT22 sensor");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};
```

Para que este bloco possa ser traduzido em código Python, foi necessário defini-lo também no arquivo “generator_stubs.js”. A variável “`value_pin`” declarada na primeira linha do método corresponde ao valor retornado pelo bloco associado, neste caso sendo um número de porta, e a variável “type” corresponde ao tipo do sensor escolhido no *dropdown*. Em seguida, são definidas as bibliotecas que devem ser importadas no início do programa através da variável “`Blockly.Python.definitions_`”. O código final é formado através da concatenação de *strings* na variável “`code`”. Nele, é inicializado um objeto na variável dhts com a classe apropriada de acordo com o tipo de sensor e no pino especificado, em seguida é feita uma medida inicial para descartar qualquer lixo que possa estar armazenado no *buffer* do sensor e, ao final, são aguardados dois segundos, respeitando o período de amostragem do sensor.

```
/// Start DHT Sensor
Blockly.Python['dht_init'] = function(block) {
 var value_pin = Blockly.Python.valueToCode(block, 'pin',
Blockly.Python.ORDER_ATOMIC);
 var type = block.getFieldValue('DHT_TYPE');
 Blockly.Python.definitions_['import_machine'] = 'import machine';
 Blockly.Python.definitions_['import_dht'] = 'import dht';
 Blockly.Python.definitions_['import_time'] = 'import time';
```

```
 var code = 'dhts=dht.' + type + '(machine.Pin(' + value_pin +
'));dhts.measure();time.sleep(2)\n';
 return code;
};
```

Após isso, ainda foi necessário atualizar as entradas desses blocos no XML da Toolbox. É interessante notar que o item “shadow” pode ser colocado dentro de cada entrada (“value”) e serve para definir uma sugestão de bloco alí esperado. Por exemplo, dentro da entrada “pin” foi definida uma sugestão de bloco do tipo “pinout”, isso faz com que este bloco seja automaticamente incluído quando o usuário criar um bloco “Start DHT11/22 Sensor”, tal como pode ser visto na Figura 12 mostrada anteriormente. A melhor parte é que isso torna a interface mais amigável e intuitiva ao mesmo tempo que não limita as opções, pois caso o usuário deseje conectar um bloco de outro tipo alí ele pode simplesmente arrastá-lo por cima da sugestão.

```
  <category name="DHT11/22 Sensor">
    <label text="DHT11/22 Temperature and Humidity Sensor"></label>
    <block type="dht_init">
      <field name="BLOCK_DHT_INIT">Start DHT11/22 sensor</field>
      <value name="pin">
        <shadow type="pinout">
          <field name="PIN"></field>
        </shadow>
      </value>
    </block>
    <block type="dht_measure">
      <field name="MSG_MEASURE_DHT">Measure DHT11/22 Sensor</field>
    </block>
    <block type="dht_read_temp">
      <field name="MSG_READ_DHT_TEMP">Read DHT11/22 Temperature</field>
    </block>
    <block type="dht_read_humidity">
      <field name="MSG_READ_DHT_HUMI">Read DHT11/22 Humidity</field>
    </block>
  </category>
```

Os demais blocos do sensor DHT seguem o mesmo padrão de implementação e seu código fonte está disponível no Apêndice A.

Por fim, os blocos DHT puderam ser adicionados através da interface do BIPES e foi gerado o código Python que está exposto abaixo. Vale lembrar que nesse exemplo os blocos de leitura de valores estavam soltos e para que o valor dos blocos com retorno seja utilizado eles devem ser encaixados em outro bloco.

```
import machine
import dht
```

```
import time

# Start DTH Sensor
dhts=dht.DHT11(machine.Pin((16)));dhts.measure();time.sleep(1)

# Measure DHT11/22 Sensor
dhts.measure()

# Read DHT11/22 Temperature
dhts.temperature()

# Read DHT11/22 Humidity
dhts.humidity()
```

3.3 MÓDULO RELÉ

Um módulo Relé (Figura 13) também foi utilizado na aplicação de validação das implementações dos módulos MQTT. Suas especificações técnicas estão disponíveis na Tabela 5.

Tabela 5 – Especificações do módulo relé

Parâmetro	**Valor**
Tipo	Digital
Circuito de Driver	Optoacoplado
Sinal de controle	Nível TTL (Lógica Negativa)
Bobina	5VDC 75mA
Carga nominal do relê	15A 125VAC , 10A 250VAC
Carga nominal do módulo	Carga nominal do módulo
Tempo de acionamento de contato	10ms

Fonte: Baú da Eletrônica[16]

[16] Disponível em: <https://www.baudaeletronica.com.br/modulo-rele-5v.html>. Acesso em: 21 de novembro de 2020.

Figura 13 – Módulo Relé

Fonte: Elaborado pelo autor.

3.3.1 Bloco para o módulo relé

Controlar esse módulo relé consiste em nada mais do que setar como alto ou baixo o pino a qual seu sinal de controle está conectado. Mesmo assim, para tornar a utilização desses dispositivos anexos mais amigável ao usuário foi desenvolvido um bloco para controle do relé, conforme a Figura 14.

Figura 14 – Bloco para controle do módulo relé

Fonte: Elaborado pelo autor.

O funcionamento do bloco baseia-se em alterar o nível da saída no pino utilizado de acordo com a necessidade de ativar ou desativar o relé, setando como baixo para ativar e alto para desativar.

Seguindo o mesmo método para adição de novos blocos utilizado no sensor DHT foi definido o bloco para controle do módulo relé. Sua implementação está disponível no Apêndice A.

3.4 MÓDULO MQTT

Nesta primeira abordagem foram desenvolvidos os blocos para comunicação MQTT que permitem uma configuração completa sobre os parâmetros de conexão e um maior controle sobre as mensagens trocadas. Portanto, este é direcionado a um usuário que tem um maior domínio sobre a tecnologia e busca mais flexibilidade. Como mencionado anteriormente, o módulo foi desenvolvido utilizando a biblioteca

umqtt do MicroPython como base.

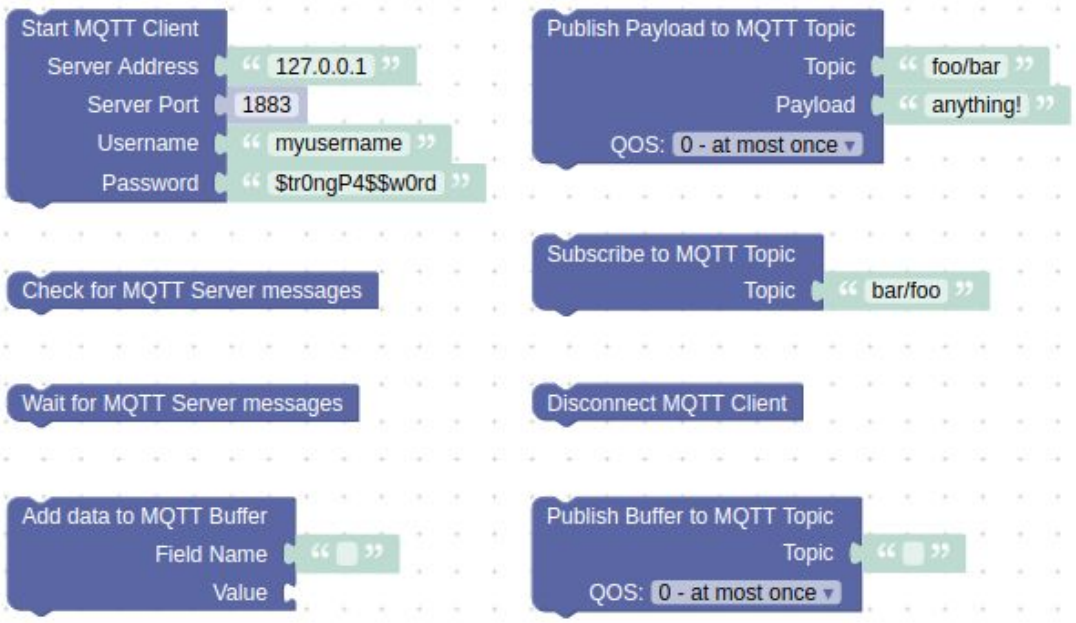
3.4.1 Blocos do módulo MQTT

Foram desenvolvidos nove blocos para o módulo MQTT. Sete deles implementam funções essenciais da biblioteca umqtt e dois oferecem funções para auxiliarem na utilização do serviço ThingSpeak, que trabalha com diversas variáveis dentro de um mesmo tópico, necessitando do envio da mensagem com payload em um formato específico. A maioria deles segue o mesmo padrão de implementação já demonstrado anteriormente e seus códigos-fonte podem ser conferidos no Apêndice A. Estes blocos, representados na Figura 15, são:

- "*Start MQTT Client*" – para inicializar a conexão com o *broker*;
    - Recebe como entradas o endereço do broker, a porta de conexão, o nome de usuário e senha utilizados para autenticação;
- "*Publish Payload to MQTT Topic*" – para publicar uma mensagem em um tópico no *broker*;
    - Recebe como entradas a mensagem (qualquer tipo de variável) e o tópico em que será publicada;
- "*Subscribe to MQTT Topic*" – para se inscrever para atualizações de um tópico no *broker*;
    - Recebe como entrada o tópico a ser subscrito;
- "*Wait for MQTT Server Messages*" – para receber mensagens do servidor sobre os tópicos subscritos de forma bloqueante;
- "*Check for MQTT Server Messages*" – para receber mensagens do servidor sobre os tópicos subscritos de forma não-bloqueante;
- "*Disconnect from MQTT Client*" – para encerrar a conexão e liberar os recursos utilizados;
- "*Add data to MQTT Buffer*" – função auxiliar para concatenar diversos campos em um mesmo tópico, seguindo o padrão de implementação do Thingspeak;
    - Recebe como entrada o dado (convertido para string) e o nome do campo a que o dado pertence;
- "*Publish Buffer to MQTT Topic*" – função auxiliar para publicar em um tópico o buffer seguindo o padrão de implementação do Thingspeak;

- Recebe como entrada o tópico de publicação;
- O payload publicado segue o formato: `field1=<value>&field2=<value>[...]`;

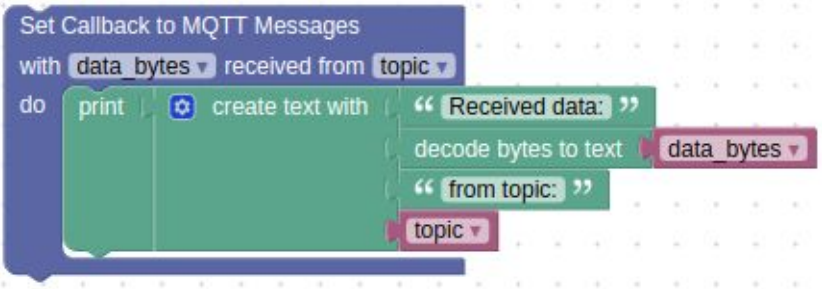
Figura 15 – Blocos de inicialização e encerramento do cliente MQTT



Fonte: Elaborado pelo autor.

O bloco restante "*Set Callback to MQTT Messages*" é um pouco diferente dos demais e, por isso, merece uma atenção especial. Este bloco define uma função que deve ser executada sempre que o cliente MQTT receber uma mensagem de um tópico subscrito. O usuário pode então programar o que ele quer fazer com a mensagem recebida, como na implementação representada na Figura 16, onde simplesmente foi impresso no console o tópico e o conteúdo recebido. É importante notar que a função de callback é a mesma para todos os tópicos subscritos, portanto isso deve ser tratado internamente no bloco caso o usuário se inscreva em mais de um tópico, por exemplo.

Figura 16 – Bloco de definição do callback para o MQTT



Fonte: Elaborado pelo autor.

Em sua definição de formato no arquivo “block_definitions.js” foram necessários dois atributos que ainda não haviam sido utilizados em outro bloco, eles são a adição de um campo do tipo “`FieldVariable`”, para determinar uma nova variável que deve ser utilizada dentro desse bloco, e a adição de uma entrada do tipo “`do`”, que provê essa estrutura de aninhamento para o encaixe de outros blocos.

```
Blockly.Blocks['mqtt_set_callback'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("Set Callback to MQTT
Messages"), "BLOCK_MQTT_SET_CALLBACK");
   this.appendDummyInput()
       .appendField('with')
       .appendField(new Blockly.FieldVariable('data_bytes'), 'MQTT_DATA_VAR')
       .appendField('received from')
       .appendField(new Blockly.FieldVariable(
         'topic',
         null,
         ['String'],
         'String'
       ), 'MQTT_TOPIC_VAR');
   this.appendStatementInput('do')
       .appendField('do');
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setInputsInline(false);
this.setTooltip("Callback function must have topic and msg parameters");
this.setHelpUrl("http://www.bipes.net.br");
 }
};
```

A definição do código responsável por traduzir o bloco para Python é um pouco mais complexa para este bloco devido a necessidade da criação de uma função para o callback com os blocos aninhados. O código que ficará dentro dessa função é obtido através do comando “`Blockly.Python.statementToCode(block, 'do')`”. A determinação de uma nova função é realizada pelo comando “`Blockly.Python.provideFunction_()`”, cujos argumentos são primeiramente o nome desejado para a função, e em seguida, cada linha como deve aparecer no código final. O comando retorna o nome que foi atribuído para a função criada, sendo sua utilização necessária para evitar conflitos caso por ventura sejam definidas 2 vezes funções com o mesmo nome. É importante notar que neste bloco foi preciso aplicar um “*fix*” baseado no código base de definição de blocos de função do Blockly, pois para que variáveis globais fora de seu escopo de execução pudessem ser acessadas dentro da função de callback elas precisavam ser

inicializadas utilizando a keyword global. O código completo para este bloco está disponível abaixo.

```
Blockly.Python['mqtt_set_callback'] = function(block) {
 var data_var_name =
Blockly.Python.variableDB_.getName(block.getFieldValue('MQTT_DATA_VAR'),
Blockly.VARIABLE_CATEGORY_NAME);
 var topic_var_name =
Blockly.Python.variableDB_.getName(block.getFieldValue('MQTT_TOPIC_VAR'),
Blockly.VARIABLE_CATEGORY_NAME);
 // Fix for global variables inside callback
 // Piece of code from generators/python/procedures.js
 // Add a 'global' statement for every variable that is not shadowed by a local
parameter.
 var globals = [];
 var varName;
 var workspace = block.workspace;
 var variables = Blockly.Variables.allUsedVarModels(workspace) || [];
 for (var i = 0, variable; variable = variables[i]; i++) {
   varName = variable.name;
   if (block.getVars().indexOf(varName) == -1 && varName != data_var_name &&
varName != topic_var_name) {
   globals.push(Blockly.Python.variableDB_.getName(varName,
     Blockly.VARIABLE_CATEGORY_NAME));
   }
 }
 // Add developer variables.
 var devVarList = Blockly.Variables.allDeveloperVariables(workspace);
 for (var i = 0; i < devVarList.length; i++) {
   globals.push(Blockly.Python.variableDB_.getName(devVarList[i],
     Blockly.Names.DEVELOPER_VARIABLE_TYPE));
 }
 globals = globals.length ? Blockly.Python.INDENT + 'global ' + globals.join(',
') : '';
 // End of code from generators/python/procedures.js

 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';

 var funct_code = Blockly.Python.statementToCode(block, 'do');
 var function_name = Blockly.Python.provideFunction_(
   'mqtt_callback',
   ['def ' + Blockly.Python.FUNCTION_NAME_PLACEHOLDER_ +
'('+topic_var_name+','+data_var_name+'):',
   globals,
   Blockly.Python.INDENT + topic_var_name + " = " + topic_var_name + ".decode()",
   funct_code]);

 var code = 'mqtt_client.set_callback(' + function_name + ')\n';
 return code;
};
```

Também foram criados alguns blocos para auxiliarem no tratamento dos dados recebidos pelo callback do MQTT. Estes blocos de implementação muito simples (Figura 17) são conversores de tipo de variável, servindo para decodificar bytes em *string*, converter uma variável em string, converter uma variável em int,

converter uma variável em float. O código-fonte está disponível no Apêndice A

Figura 17 – Blocos auxiliares para tratamento de dados

Fonte: Elaborado pelo autor.

3.5 MÓDULO EASYMQTT

Esta segunda abordagem parte do pressuposto de oferecer um método ainda mais fácil de prototipar aplicações IoT utilizando o MQTT, em que o usuário não precisasse criar um conta, configurar parâmetros ou entender a fundo sobre o tratamento de dados binários recebidos por *callback*. Para isso, foi necessário criar uma infraestrutura para esse serviço no BIPES.

Uma suposição que foi tomada para o desenvolvimento de EasyMQTT é a de que existiria um controle por sessões para isolar tópicos de uma aplicação de outras. Essa sessão nada mais seria do que o primeiro nível da estrutura de tópicos do MQTT. Por exemplo, imaginando uma sessão com o nome “my_session”, caso o usuário desejasse enviar dados para os tópicos “temperature” e “humidity”, na verdade, a implementação do EasyMQTT estaria utilizando os tópicos “my_session/temeprature” e “my_session/humidity”, existindo uma camada de abstração para tratar disto.

3.5.1 Configuração do Broker MQTT Mosquitto

Como o BIPES já dispunha de um servidor na nuvem, o mesmo foi utilizado para a instalação e configuração das ferramentas necessárias. O primeiro passo foi providenciar um servidor MQTT e para tal foi escolhido o projeto Eclipse Mosquitto, um dos mais utilizados para tal função, além de ser *open-source,* leve e com uma boa documentação.

O servidor do BIPES está localizado da Google Cloud e roda a distribuição Linux Debian 10. Estando já conectado via SSH ao servidor do BIPES, foi

executado no terminal o seguinte comando para instalar o mosquitto:

```
sudo apt install mosquitto
```

Em seguida, foi necessário configurar um usuário e senha de acesso ao *broker*. Para isso foi executado o seguinte comando, sendo bipes o nome de usuário. Foi necessário inserir a senha desejada duas vezes quando perguntado.

```
sudo mosquitto_passwd -c /etc/mosquitto/paswd bipes
```

Então, para concluir a configuração, foi necessário abrir em um editor de texto com permissões de superusuário arquivo "`/etc/mosquitto/mosquitto.conf`" e adicionar as seguintes linhas ao final:

```
   allow_anonymous false
   password_file /etc/mosquitto/passwd
```

Por fim, o servidor Mosquitto foi iniciado através do comando:

```
sudo systemctl start mosquitto
```

3.5.2 Configuração do banco de dados MongoDB

Foi utilizado um banco de dados não relacional para a implementação do EasyMQTT por ser possível fazer uma boa analogia entre a estrutura planejada para o EasyMQTT e o modelo de dados base do MongoDB e pela flexibilidade quanto aos dados armazenados no banco. Pensando no modelo de dados, cada sessão EasyMQTT será um banco de dados, cada tópico uma coleção e cada mensagem um documento (Figura 18). Assim, é fácil armazenar e recuperar os dados recebidos via MQTT.

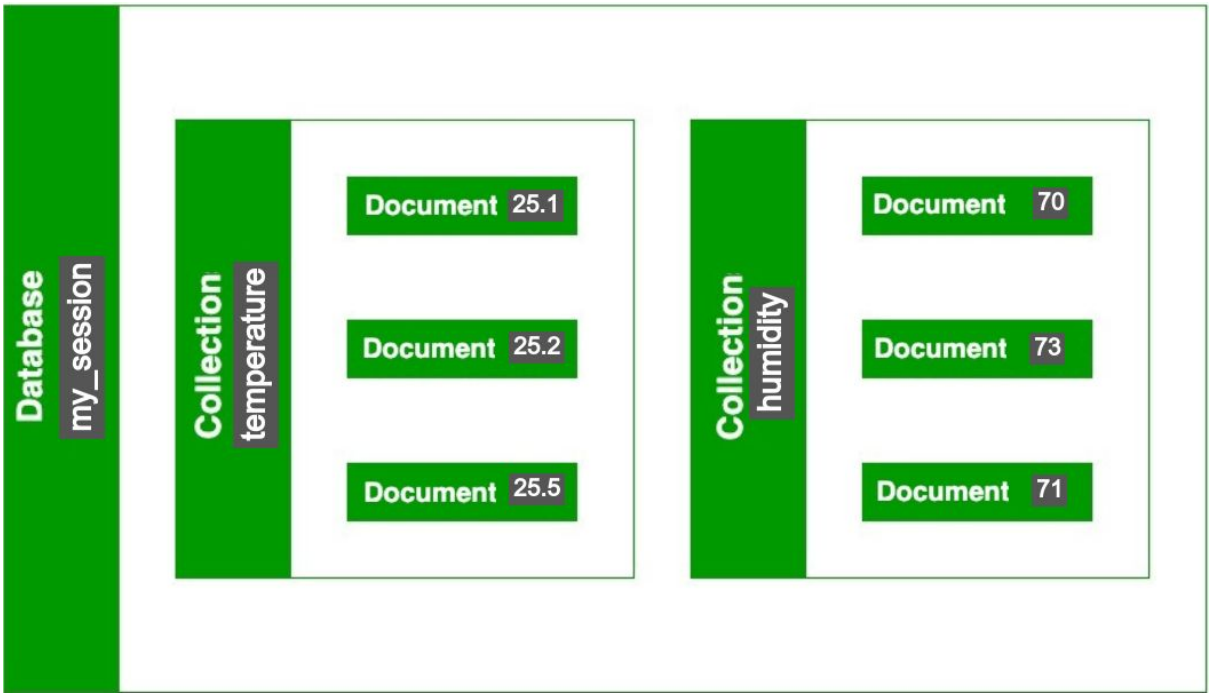
Para instalar o MongoDB foi executado os seguintes comandos no terminal:

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -
echo "deb http://repo.mongodb.org/apt/debian buster/mongodb-org/4.4 main" | sudo
tee /etc/apt/sources.list.d/mongodb-org-4.4.list
sudo apt update
sudo apt install mongodb-org
```

Depois, só foi necessário iniciá-lo através do comando:

```
sudo systemctl start mongod
```

Figura 18 – Modelo de dados do EasyMQTT no MongoDB



Fonte: Adaptado de GeeksforGeeks[17].

3.5.3 Cliente MQTT em Python

A fim de realizar o armazenamento das mensagens enviadas ao *broker* MQTT do BIPES no banco de dados MongoDB foi desenvolvido um script Python que é executado como um serviço no servidor. Ele usa as bibliotecas paho.mqtt[18], também da Eclipse Foundation, e pymongo[19], também mantida pela MongoDB, para se conectar ao *broker* e banco de dados, respectivamente.

O código abaixo é de funcionamento bastante simples: utilizando as bibliotecas mencionadas, são abertas as conexões com os servidores e então é definido que o cliente MQTT deverá se subscrever a mensagens de qualquer tópico setando o tópico de destino como o caractere curinga “#”. Ao recebimento de cada mensagem é executado o *callback* definido, que primeiro verifica se o nome tópico está no formato apropriado “sessão/tópico” e, no caso positivo, armazena o dado recebido no banco de dados junto a uma timestamp do momento do recebimento.

[17] Disponível em: <https://www.geeksforgeeks.org/mongodb-database-collection-and-document/>. Acesso em: 21 de novembro de 2020.
[18] Disponível em: <https://www.eclipse.org/paho/>. Acesso em: 21 de novembro de 2020.
[19] Disponível em: <https://api.mongodb.com/python/current/>. Acesso em: 21 de novembro de 2020.

```
from pymongo import MongoClient
import paho.mqtt.client as mqtt
import time

mongo_client = MongoClient('127.0.0.1', 27017)
mqtt_client = mqtt.Client()

def mqtt_on_connect(client, userdata, flags, rc):
   client.subscribe("#")


def mqtt_on_message(client, userdata, msg):
   full_topic = msg.topic.split("/", 1)

   if len(full_topic) < 2:
       print("Invalid Topic")
       return

   session = full_topic[0]
   topic = full_topic[1]
   data = msg.payload.decode()
   database = mongo_client[session]
   collection = database[topic]
   collection.insert_one({"data": data, "timestamp": int(time.time())})
   print("Session:", session, "Topic:", topic, "Data:", data)


mqtt_client.on_connect = mqtt_on_connect
mqtt_client.on_message = mqtt_on_message
mqtt_client.username_pw_set("bipes", password="m8YLUr5uW3T")
mqtt_client.connect("127.0.0.1", 1883, 60)
mqtt_client.loop_forever()
```

Para que publicações para o *broker* MQTT possam ser feitas através da API implementada na próxima sessão foi criado também um script publicador de funcionamento simples. O “publish.py”, disponível abaixo, recebe como argumentos o tópico e a mensagem que deve ser publicada e se tudo correr bem encerra retornando 0.

```
import paho.mqtt.client as mqtt
import sys

if len(sys.argv) != 3:
   print("Usage: publish.py topic value")
   exit(1)

mqtt_client = mqtt.Client()
mqtt_client.username_pw_set("bipes", password="m8YLUr5uW3T")
mqtt_client.connect("bipes.net.br", 1883, 60)
mqtt_client.publish(sys.argv[1], sys.argv[2]);
mqtt_client.disconnect();
exit(0)
```

Para que os scripts pudessem ser executados foi preciso instalar as

bibliotecas necessárias no servidor utilizando o “pip”. Para tal foram usados os seguintes comandos:

```
sudo apt install python3-pip
/usr/bin/python3 -m pip install --upgrade pip
/usr/bin/python3 -m pip install pymongo
/usr/bin/python3 -m pip install paho.mqtt
```

Depois, a fim de tornar o script “subscriber.py” um serviço, foi criado o arquivo “/etc/systemd/system/easymqtt.service” com seguinte conteúdo:

```
[Unit]
Description=EasyMQTT Subscriber Service
Requires=mongod.service
Requires=mosquitto.service

[Service]
WorkingDirectory=/opt/easymqtt/
Type=idle
ExecStart=/usr/bin/python3 /opt/easymqtt/subscriber.py &> /dev/null
Restart=always

[Install]
WantedBy=multi-user.target
```

Por fim, a inicialização do serviço subscritor foi realizada com os comandos:

```
sudo systemctl enable easymqtt
sudo systemctl start easymqtt
```

3.5.4 API para visualização dos dados

Tendo todas as mensagens recebidas pelo broker do EasyMQTT armazenadas no banco de dados, foi necessário desenvolver uma forma de acessá-las e torná-las disponíveis ao usuário. Assim, como o servidor do BIPES já contava com um servidor HTTP Apache[20] configurado e servindo algumas páginas em PHP, foi escolhida tal linguagem para implementação da API.

Primeiramente, foi necessário instalar a biblioteca para permitir a conexão ao banco de dados MongoDB a partir do PHP. Para isso foram executados os comandos:

```
sudo apt install php-pear php-dev
sudo pecl install mongodb
```

[20] Disponível em: <https://httpd.apache.org/>. Acesso em 22 de novembro de 2020.

Após, foi adicionada a linha abaixo aos arquivos "`/etc/php/7.3/apache2/php.ini`" e "`/etc/php/7.3/cli/php.ini`".

```
extension=mongo.so
```

Então, foi reiniciado o servidor HTTP Apache para que as alterações no PHP entrassem em vigor.

```
sudo systemctl restart apache2
```

No servidor do BIPES, os arquivos servidos pelo Apache2 estão localizados na pasta "`/var/www/html/`". Portanto, para a API do EasyMQTT foi criada a pasta "`/var/www/html/easymqtt`". Dentro dessa pasta foram executados os seguintes comandos para configurar a biblioteca do MongoDB:

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') ===
'795f976fe0ebd8b75f26a6dd68f78fd3453ce79f32ecb33e7fd087d39bfeb978342fb73ac986cd4f
54edd0dc902601dc') { echo 'Installer verified'; } else { echo 'Installer
corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"
php composer.phar require mongodb/mongodb
```

A API foi estruturada tendo quatro funções diferentes, todas sendo acessadas através de requisições do tipo GET e retornando dados em formato JSON. As funções "listsessions", "getsession", "gettopic", "publish" e "clearsession" estão melhor detalhadas nas Tabelas 6, 7, 8, 9 e 10, respectivamente. O código-fonte de todas as funções da API está disponível no Apêndice B.

Tabela 6 – Especificação da função "listsessions" da API

Parâmetro	**Descrição**
Função	listsessions
URL	http://bipes.net.br/easymqtt/listsessions.php
Método	GET
Retorno	JSON `{"success":true,"result":[{"session":"session_id","topicsCount":1,"messagesCount":2}]}`

Fonte: Elaborado pelo autor.

Tabela 7 – Especificação da função “getsession” da API

Parâmetro	**Descrição**
Função	getsession
URL	http://bipes.net.br/easymqtt/getsession.php
Método	GET
Parâmetros	session=<session_id>
Retorno	JSON `{"success":true,"result":["umidade","temperatura"]}`

Erros	Argumentos inválidos:
	`{"success":false,"result":"Invalid Parameters"}`
	Sessão vazia:
	`{"success":false,"result":"Session 'clean_session' does not contain any topic"}`

Fonte: Elaborado pelo autor.

Tabela 8 – Especificação da função “gettopic” da API

Parâmetro	**Descrição**
Função	gettopic
URL	http://bipes.net.br/easymqtt/gettopic.php
Método	GET
Parâmetros	session=<session_id>&topic=<topic_name>&since=<timestamp [opcional]>
Retorno	JSON `{"success":true,"result":[{"timestamp":1605998938,"data":"0"}]}`
Erros	Argumentos inválidos: `{"success":false,"result":"Invalid Parameters"}`

Fonte: Elaborado pelo autor.

Tabela 9 – Especificação da função “publish” da API

Parâmetro	**Descrição**
Função	publish
URL	http://bipes.net.br/easymqtt/publish.php
Método	GET
Parâmetros	session=<session_id>&topic=<topic_name>&value=<value>
Retorno	JSON `{"success":true,"result":"Value '1' published to topic 'test' successfully!"}`

Erros	Argumentos inválidos:
	`{"success":false,"result":"Invalid Parameters"}`
	Valor não numérico:
	`{"success":false,"result":"Error publishing value 'a' to topic 'test'. Non-numeric input value!"}`
	Erro na publicação MQTT:
	`{"success":false,"result":"Error publishing value '1' to topic 'test'. <some other error here>"}`

Fonte: Elaborado pelo autor.

Tabela 10 – Especificação da função “clearsession” da API

Parâmetro	**Descrição**
Função	clearsession
URL	http://bipes.net.br/easymqtt/clearsession.php
Método	GET
Parâmetros	session=<session_id>
Retorno	JSON `{"success":true,"result":"Session 'test' cleaned"}`
Erros	Argumentos inválidos: `{"success":false,"result":"Invalid Parameters"}`

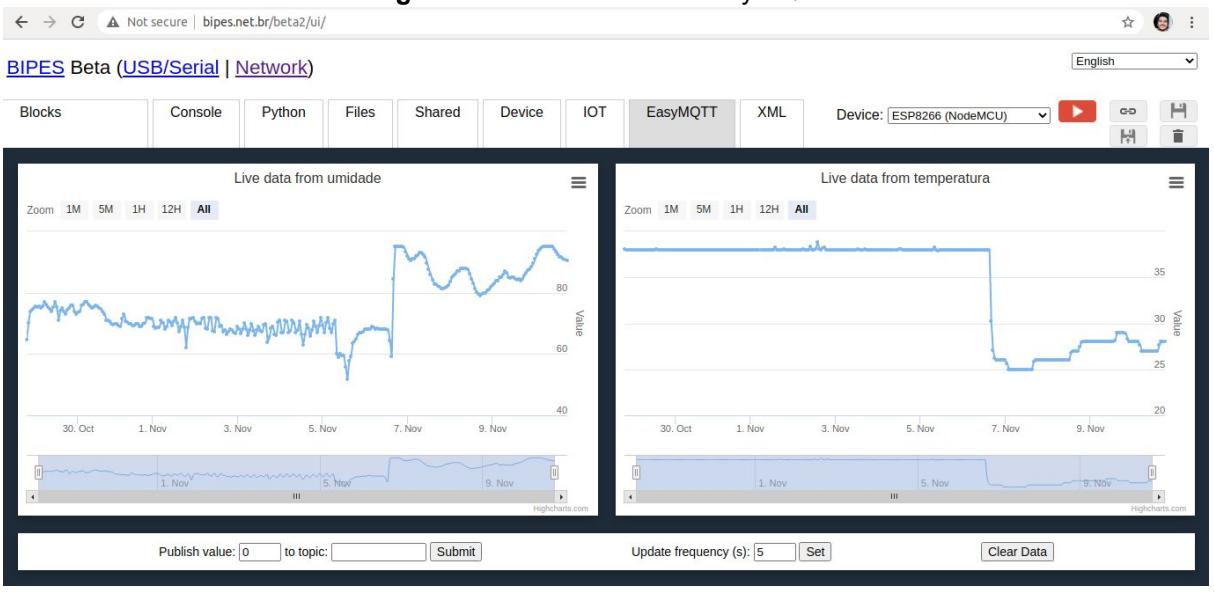
Fonte: Elaborado pelo autor.

3.5.5 Dashboard do EasyMQTT

Visando permitir uma fácil visualização dos dados enviados utilizando o EasyMQTT foi criado uma Dashboard integrada à IDE do BIPES. Esta dashboard,

vista na Figura 19, adquire a configuração de sessão automaticamente através do bloco de Inicialização do MQTT que será descrito na seção a seguir e também exibe automaticamente dados de qualquer tópico que venha a ser utilizado dentro dessa sessão, não necessitando de nenhuma configuração por parte do usuário.

Figura 19 – Dashboard do EasyMQTT



Fonte: Elaborado pelo autor.

A dashboard também pode ser acessada fora da IDE do BIPES através do endereço http://bipes.net.br/easymqtt/?session=<session_id>. Os gráficos exibidos são atualizados automaticamente em uma frequência configurável na interface. Também é possível publicar valores em qualquer tópico e limpar todos os dados na sessão através da dashboard.

A dashboard foi escrita totalmente em Javascript e utiliza para os gráficos a biblioteca Highcharts[21], gratuita para uso não comercial. O código-fonte está disponível no Apêndice C.

O EasyMQTT é uma página WEB separada, logo, foi utilizada como um iframe para ser integrada dentro da IDE do BIPES. O clique no botão EasyMQTT na lista de guias da IDE é responsável por atualizar o id da sessão utilizada dentro da dashboard. O código adicionado o arquivo “index.html” do BIPES para tal funcionalidade está ilustrado abaixo:

```
<td id="tab_mqtt" class="taboff"
onclick='document.getElementById("easymqtt_iframe").contentWindow.startEasyMQTT(w
```

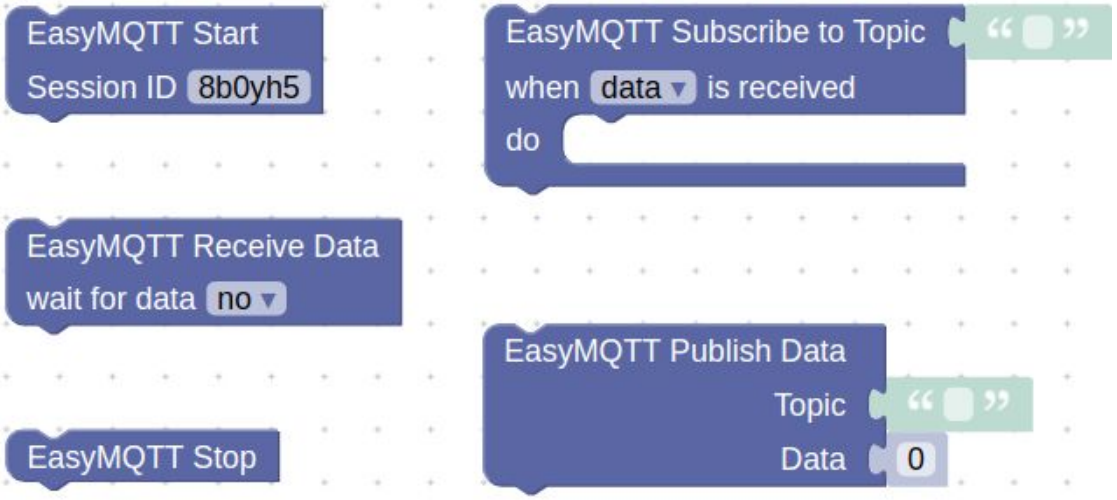
[21] Disponível em: <https://www.highcharts.com/>. Acesso em: 22 de novembro de 2020.

```
indow.easyMQTT_session);'>EasyMQTT</td>
[...]
<div id="content_mqtt" class="content" style="padding: 0;">
 <iframe id="easymqtt_iframe" src="../easymqtt/index.html" width="100%"
height="100%" frameBorder="0"></iframe>
</div>
```

3.5.6 Blocos do módulo EasyMQTT

Utilizando da mesma metodologia que para os blocos do módulo MQTT, foram desenvolvidos cinco blocos para o EasyMQTT (Figura 20). Também baseados na biblioteca umqtt, estes blocos oferecem uma implementação mais simplificada e amigável, com menos configurações necessárias. De fato, para publicar um valor em um tópico é necessário somente arrastar para o programa um bloco do tipo “EasyMQTT Start”, depois um “EasyMQTT Publish Data”, preencher o tópico e dado a ser publicado e nada mais.

Figura 20 – Blocos desenvolvidos para o EasyMQTT



Fonte: Elaborado pelo autor.

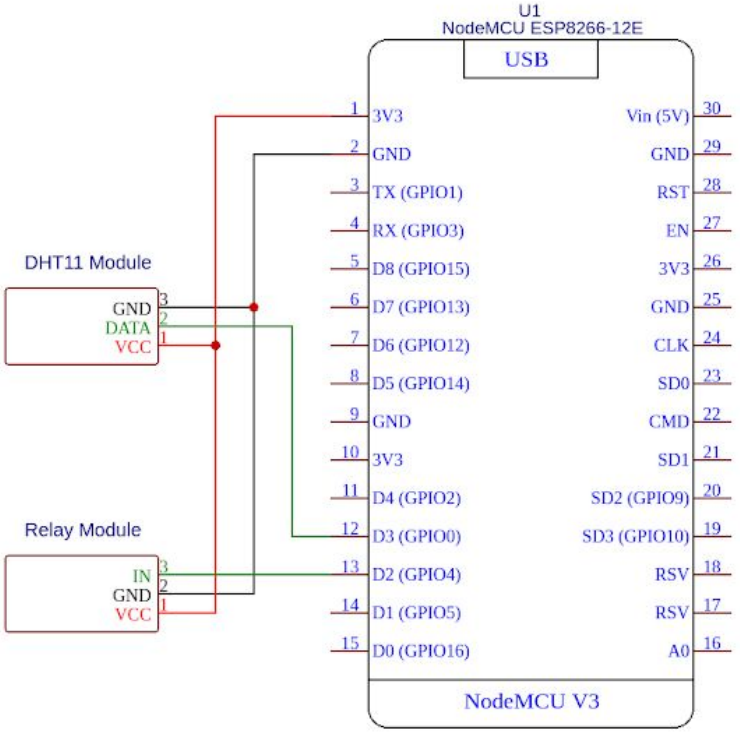
Os blocos possuem implementação muito similar aos do módulo MQTT. Uma diferença é no bloco “EasyMQTT Receive data” que engloba as funções “check” e “wait” do umqtt em um só bloco com um seletor *dropdown* para escolha do comportamento bloqueante ou não bloqueante. As maiores diferenças estão no bloco de inicialização “EasyMQTT Start”, onde foi criada uma função de geração de um id de sessão aleatório a cada inserção de um novo bloco, a gravação deste id numa variável global para utilização pela Dashboard e o *hardcoding* das configurações do servidor. O código fonte de todos os blocos está no Apêndice A.

4 RESULTADOS E DISCUSSÃO

Todo o código contribuído ao projeto BIPES está disponível em seu repositório no GitHub[22].

Com o intuito de validar os módulos desenvolvidos para o BIPES foram criadas duas aplicações de teste para resolver uma necessidade real do FITOTEC (Laboratório de Tecnologia Farmacêutica em Fitoprodutos) da UNESP – campus Assis, uma delas usando o módulo MQTT e o outra o EasyMQTT. É necessário monitorar durante alguns dias a temperatura e umidade internas de uma chocadeira de ovos sempre que novos ovos são recebidos, funcionalidade facilmente implementável com o BIPES com uma ESP8266 e sensor DHT11. Para essa necessidade foi preciso somente o uso do módulo ESP8266 sendo o publicador de dados, porém, para fins de validação, também foi adicionado um módulo relay à arquitetura para teste da função de subscrição. O diagrama esquemático do circuito montado está disponível na Figura 21.

Figura 21 – Diagrama esquemático do circuito



Fonte: Elaborado pelo autor.

O circuito foi então instalado na chocadeira de ovos presente dentro do laboratório FITOTEC na Unesp (Figura 22). O sensor foi fixado na parte interna da

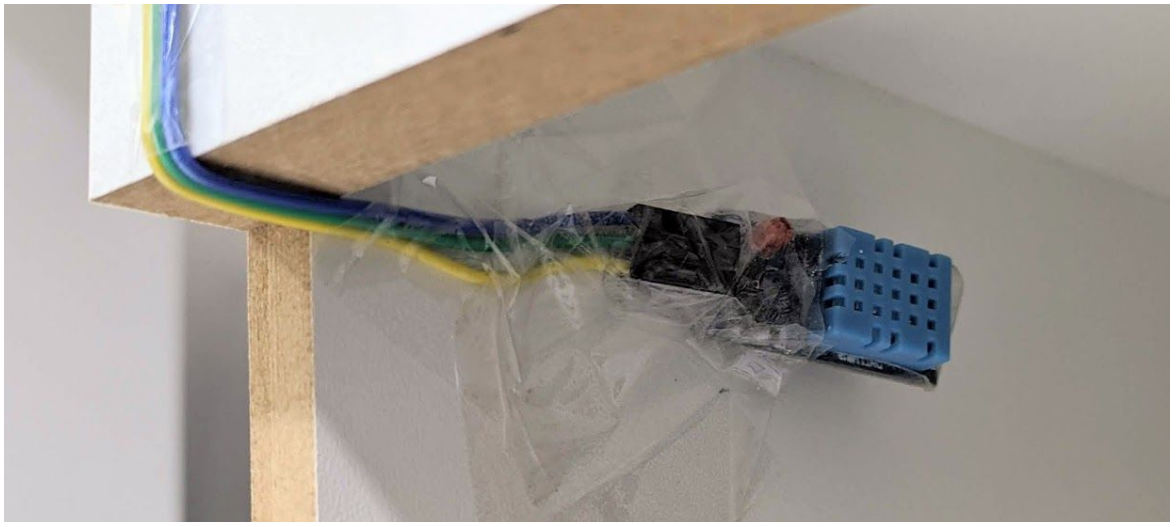
[22] Disponível em: <https://github.com/rafaelaroca/BIPES_ui_testing>. Acesso em: 26 nov. 2020.

chocadeira, conforme a Figura 23. A alimentação da placa foi realizada através de uma fonte USB genérica 5V 1.2A.

Figura 22 – Chocadeira de ovos com o módulo ESP8266 instalado

Fonte: Elaborado pelo autor.

Figura 23 – Sensor DHT11 no interior da chocadeira de ovos



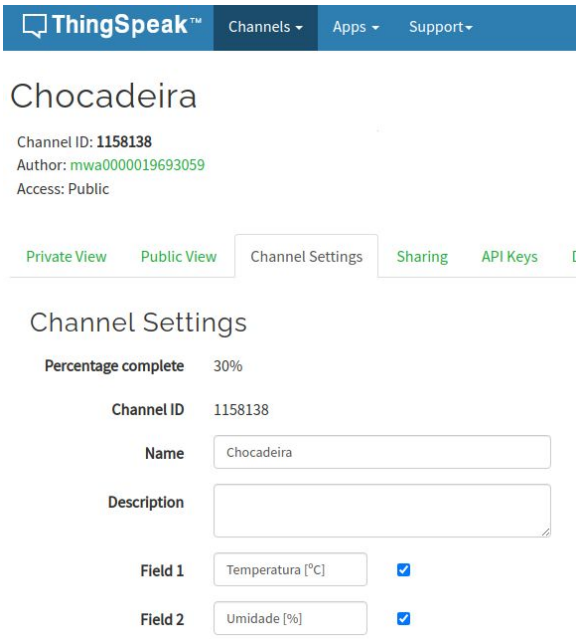
Fonte: Elaborado pelo autor.

4.1 VALIDAÇÃO DO MÓDULO MQTT

Neste primeiro teste foi utilizada a plataforma ThingSpeak como *broker* MQTT, escolha essa devido a facilidade de prototipação e visualização dos dados por ela proporcionada. Após o registro no site, foram criados dois tópicos, um

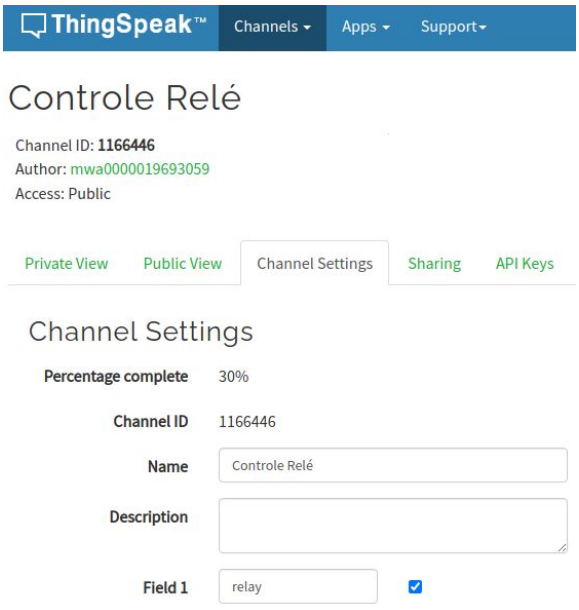
chamado de “Chocadeira” para armazenar os dados de temperatura e umidade e outro chamado de “Controle Relé” para controlar o módulo relé. O ThingSpeak apresenta uma estrutura um pouco diferente contendo campos dentro dos tópicos, assim foram definidos os campos “field1” – “Temperatura [ºC]” e “field2” – “Umidade [ºC]” dentro do tópico Chocadeira e só o campo “field1” – “relay” no tópico “Controle Relé”. A configuração dos dois tópicos no site está representada nas Figuras 24 e 25.

Figura 24 – Tópico “Chocadeira”



Fonte: Elaborado pelo autor.

Figura 25 – Tópico “Controle Relé”



Fonte: Elaborado pelo autor.

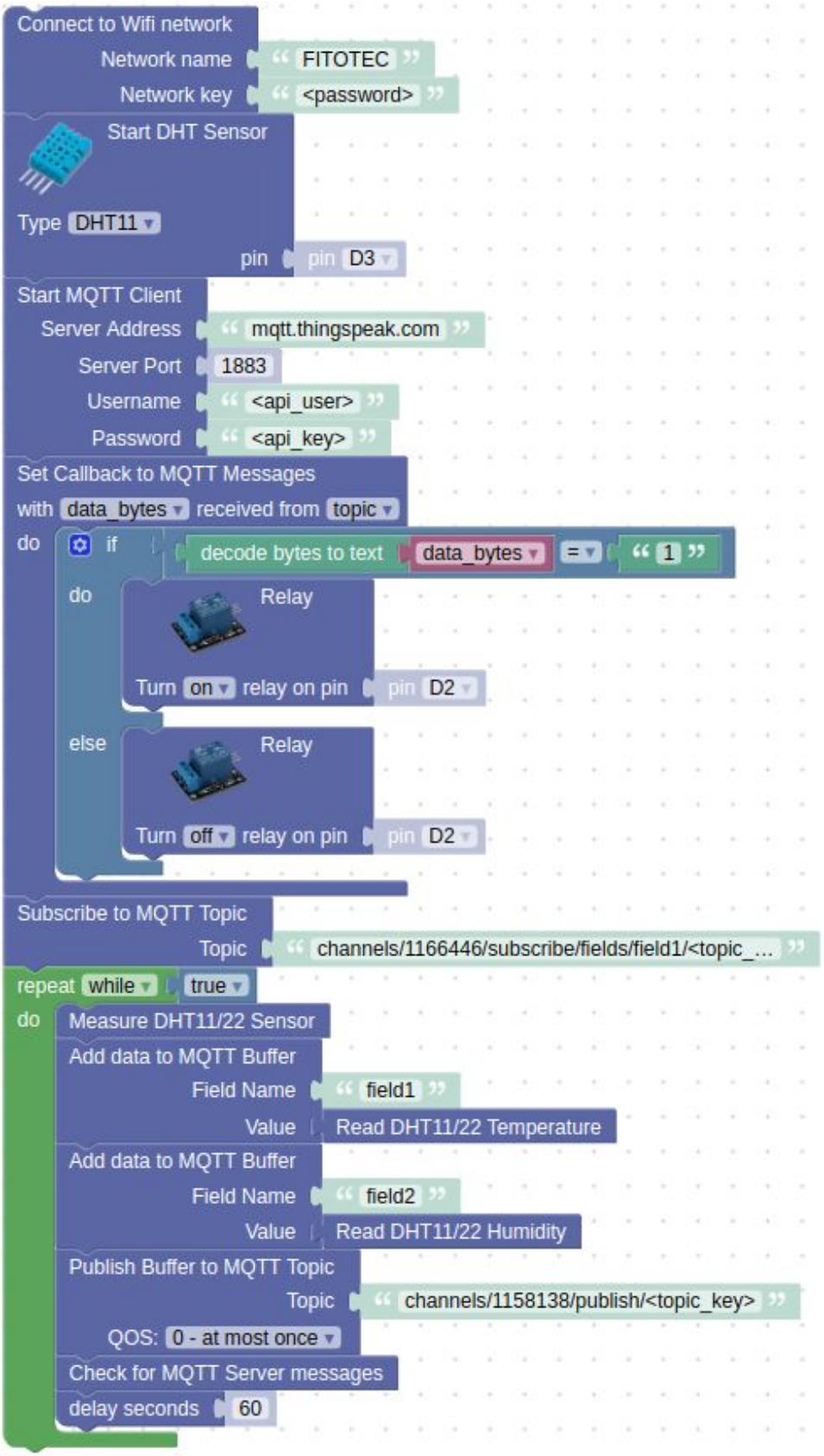
Assim, foi desenvolvido o programa em blocos no BIPES (Figura 26). O programa primeiro se conecta à rede Wi-Fi do laboratório, inicializa o sensor DHT e o cliente MQTT. Depois, é definido o que deve ser feito ao receber uma mensagem subscrita, como o cliente irá se subscrever somente a um tópico, não foi feita nenhuma verificação sobre o tópico recebido nesse caso. Quando o valor “1” for recebido o relé será ativado, caso contrário será desativado. O próximo bloco realiza a subscrição do cliente ao campo “field1” do tópico “Controle Relé”.

O *loop while,* em seguida, contém a parte principal do programa que, a cada 60 segundos, efetua a medida do sensor DHT e publica os valores medidos para o tópico “Chocadeira”. Também é efetuada uma verificação não bloqueante para possíveis mensagens do tópico subscrito.

O nível de QoS 0 foi escolhido para a publicação de mensagens MQTT pois é

o único nível suportado pelo ThingSpeak.

Figura 26 – Programa desenvolvido com o BIPES utilizando o módulo MQTT



Fonte: Elaborado pelo autor.

O código Python gerado pelo programa gerado pelo BIPES, a partir dos blocos, pode ser conferido no Apêndice D.

Para que não fosse necessário rodar sempre o programa conectando a placa à interface do BIPES e realizando a execução, o código gerado foi salvo dentro da própria placa de forma que fosse executado automaticamente após o boot. Para tal, foi utilizada a aba “Files” na interface do BIPES, então clicado no botão “Blocks to editable Python”, digitado o nome de arquivo “main.py” e clicado no botão “save”, conforme ilustrado pela Figura 27.

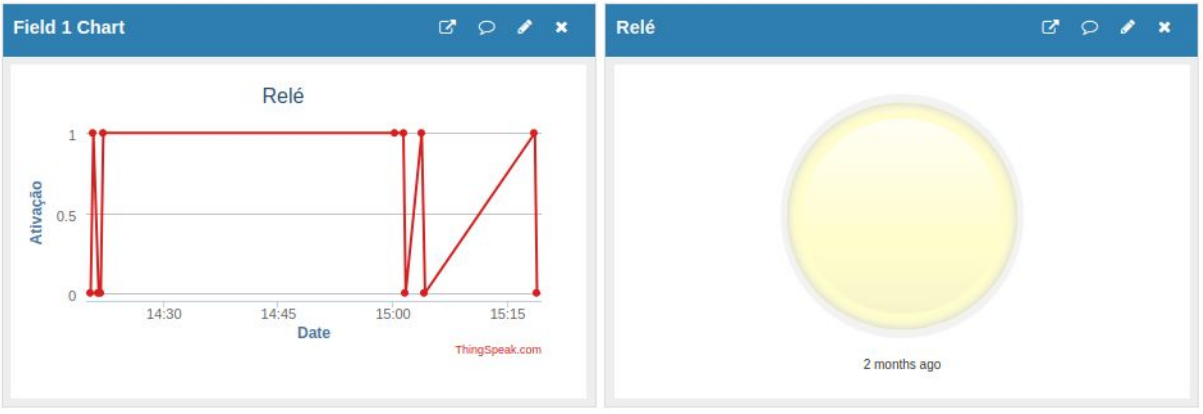
Figura 27 – Processo de salvamento do código na memória interna do ESP8266

Fonte: Elaborado pelo autor.

O histórico de dados no tópico usado para o controle do relé no ThingSpeak[23] está representado na Figura 28. Para alterar seu estado foi utilizado o seguinte link da API do ThinkSpeak:

```
GET https://api.thingspeak.com/update?api_key=JWFS8XS4KNN87QE2&field1=<0 ou 1>
```

Figura 28 – Dados coletados via MQTT utilizando o Thingspeak



Fonte: Elaborado pelo autor.

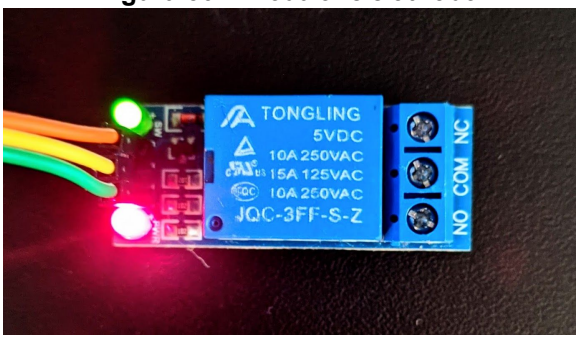
[23] Disponível em: <https://thingspeak.com/channels/1166446>. Acesso em: 23 de novembro de 2020.

Foi possível observar que o relé foi ativado e desativado exatamente conforme comandado. As Figuras 29 e 30 mostram o relé nos seus dois estados.

Figura 29 – Módulo relé desativado

Fonte: Elaborado pelo autor.

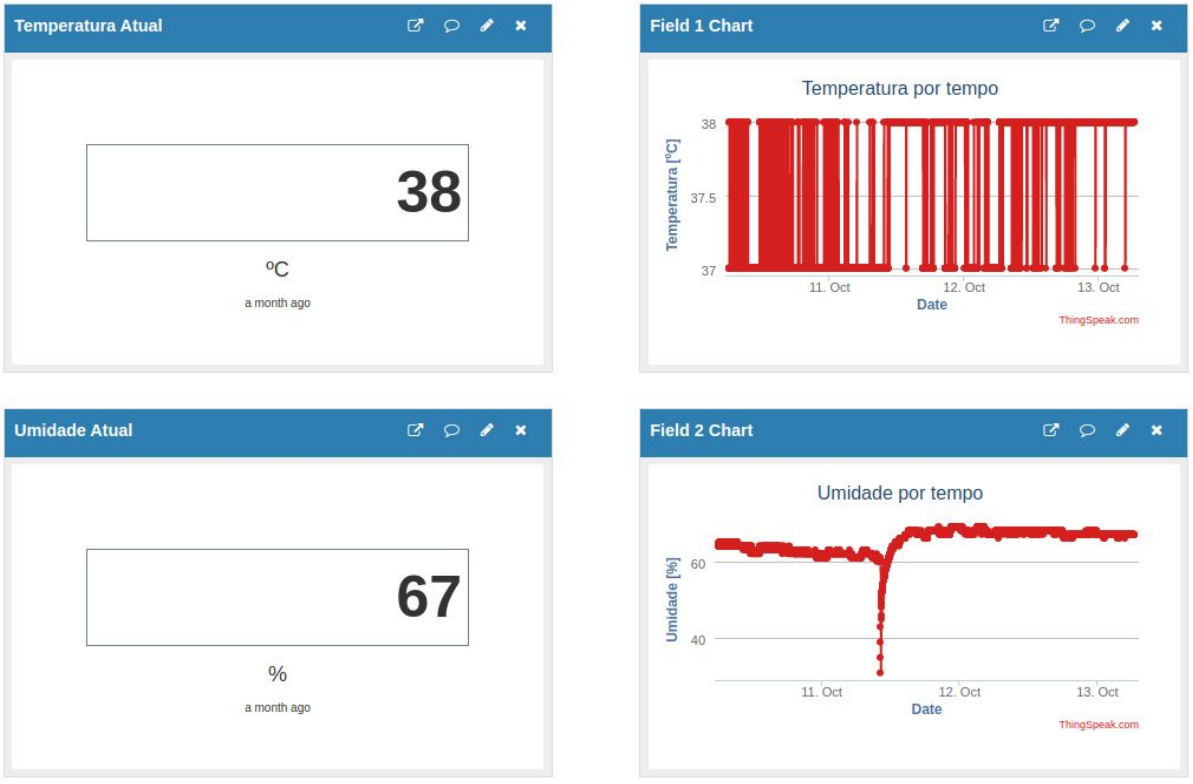
Figura 30 – Módulo relé ativado



Fonte: Elaborado pelo autor.

O programa monitorou a chocadeira de ovos durante sete dias neste primeiro teste. A Figura 31 mostra o dashboard do Thingspeak exibindo os dados do tópico da chocadeira[24] que eram atualizados em tempo real.

Figura 31 – Dados coletados via MQTT utilizando o Thingspeak



Fonte: Elaborado pelo autor.

Analisando os dados armazenados, foi possível constatar que dentre as 9293 mensagens recebidas pelo ThingSpeak no período analisado, somente 12, ou

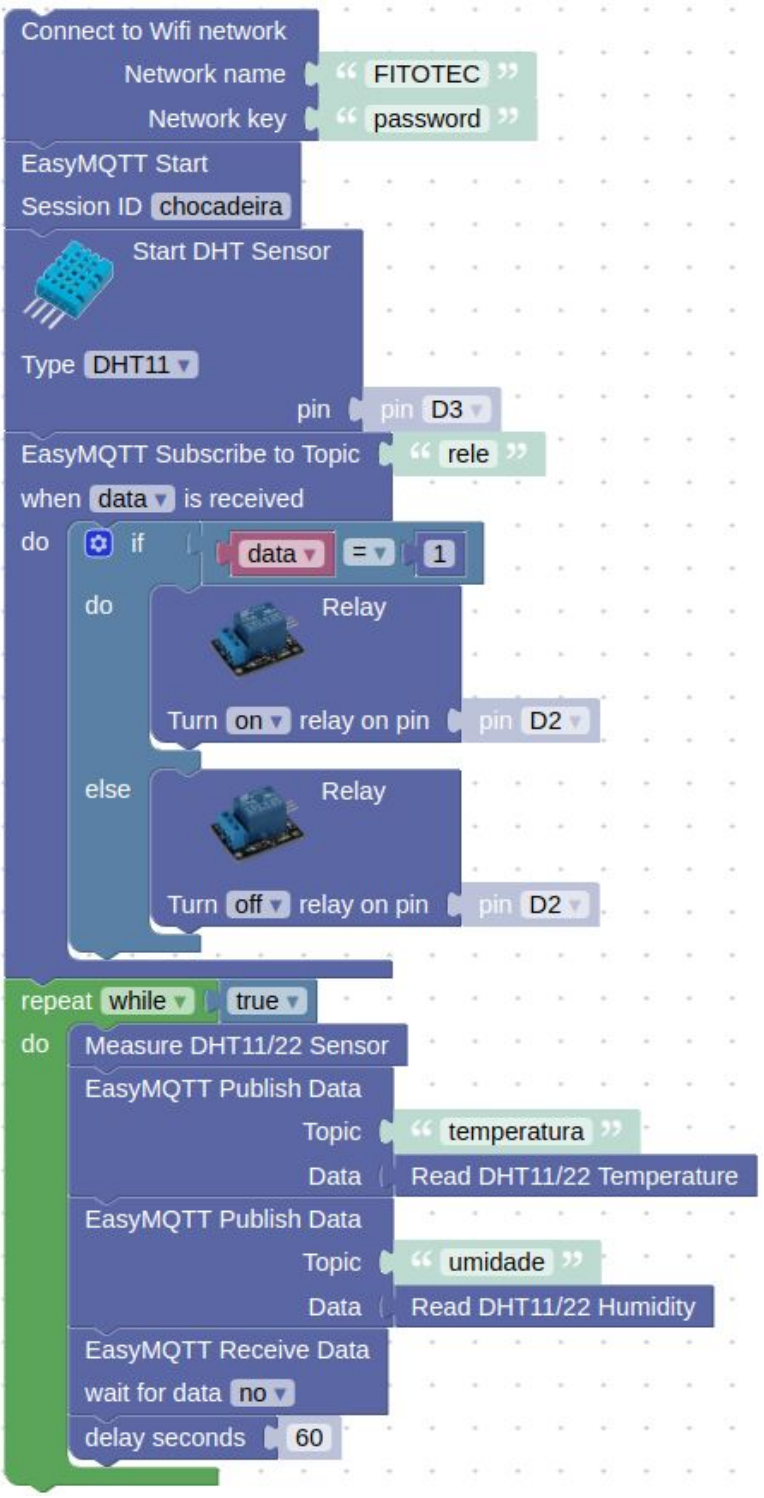
[24] Disponível em: <https://thingspeak.com/channels/1158138>. Acesso em: 23 de novembro de 2020.

0,13%, sofreram algum atraso na entrega, ou seja, o intervalo entre duas mensagens consecutivas foi maior do que 60 ± 5 segundos. A soma de todos esses atrasos corresponde a somente 320 segundos sem atualizações, o que dentro das 155 horas e 45 minutos de operação correspondem a somente 0,06% do tempo, ou seja, o uptime foi de 99,94%. A possível causa dessas pequenas anomalias parece ter sido um atraso para o *broker* do ThingSpeak processar as informações ou pequenas instabilidades de conexão, visto que no pior caso o atraso foi de 75 segundos. Os resultados mostram portanto que o programa e blocos projetados cumpriram seu papel de acordo com o esperado.

4.2 VALIDAÇÃO DO MÓDULO EASYMQTT

Para o segundo teste foi criado o programa em blocos que utiliza o módulo EasyMQTT, ilustrado na Figura 32. O código fonte Python gerado automaticamente está disponível no Apêndice E.

Figura 32 – Blocos desenvolvidos para o EasyMQTT

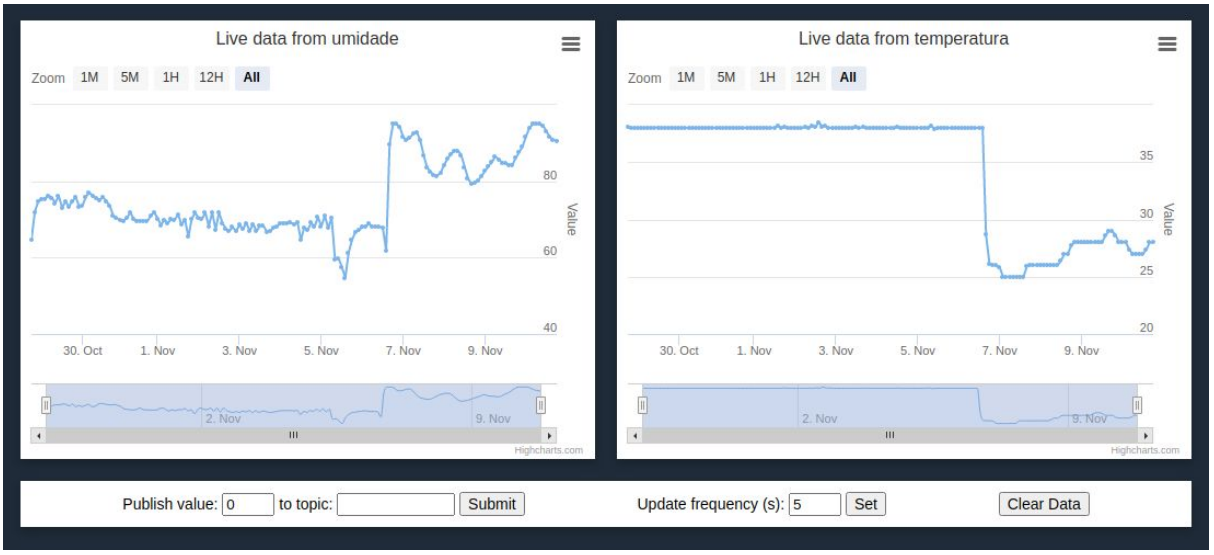


Fonte: Elaborado pelo autor.

Para que não fosse necessário sempre abrir a interface WEB do BIPES para rodar o programa na placa, o código gerado também foi salvo dentro da ESP8266, seguindo o mesmo método ilustrado pela Figura 27.

Neste teste, o programa monitorou a chocadeira de ovos durante treze dias, sendo dez dias com ela ligada e três com ela desligada. Os dados, mostrados na Dashboard na Figura 33 eram atualizados em tempo real na sessão “chocadeira” do EasyMQTT[25].

Figura 33 – Dashboard do EasyMQTT exibindo os dados coletados



Fonte: Elaborado pelo autor.

A *dashboard* do EasyMQTT permite a extração dos dados de cada gráfico, em formato JSON e CSV. Analisando os dados foi possível constatar que de 17984 mensagens recebidas pelo broker, somente 19, ou 0,11%, sofreram algum atraso na entrega. Se analisarmos o uptime, das 311 horas e 40 minutos que a placa ficou ligada, por 11 horas e 14 minutos ela ficou sem se comunicar, representando um uptime de 96,4%. O resultado ficou um pouco aquém do esperado, visto que no teste anterior foi obtido um uptime de quase 100%, porém analisando os dados pôde-se perceber que nessas poucas falhas a placa ficou por grandes períodos de tempo seguidos sem se comunicar, chegando a quase 5 horas em um deles. Também, todos os períodos de falha ocorreram no mesmo dia, 30 de outubro de 2020, o que indica que tais problemas provavelmente foram causados por instabilidades na rede da UNESP a qual a placa estava conectada.

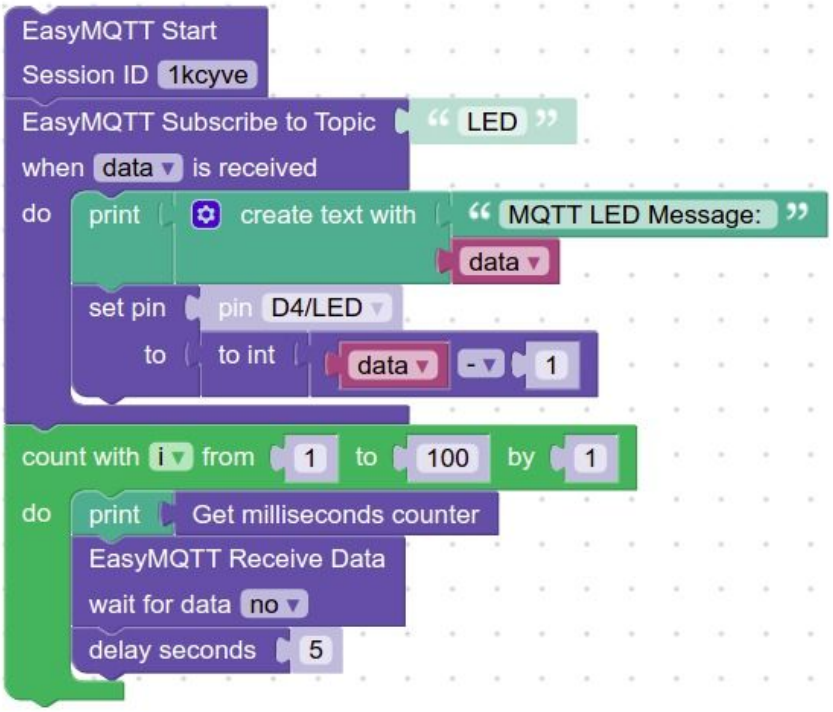
[25] Disponível em: <http://bipes.net.br/easymqtt/?session=chocadeira>. Acesso em: 23 de novembro de 2020.

Apesar disso, os blocos provaram-se funcionais e a aplicação mostrou-se proficiente em se recuperar de instabilidades na rede, pois retomou a operação com sucesso assim que as condições foram estabilizadas. Também é interessante observar que fora deste período de quedas do dia 30 de outubro, não ocorreram quaisquer outros atrasos ou falhas de conexão, sendo todas as mensagens entregues ao broker no tempo esperado de 60 ± 5 segundos.

4.3 UTILIZAÇÃO DO EASYMQTT EM MINICURSO

Além das validações propostas neste trabalho, o módulo EasyMQTT também já foi utilizado em um Minicurso sobre Sistemas Embarcados e Internet das Coisas utilizando o BIPES ministrado pelo Prof. Dr. Rafael V. Aroca. Em um dos exemplos apresentados no curso foi desenvolvido o programa abaixo (Figura 34) que utiliza um tópico do EasyMQTT para controlar um LED.

Figura 34 – Programa desenvolvido no minicurso



Fonte: Cedido pelo Prof. Dr. Rafael V. Aroca.

Para controlar esse LED, além de utilizar as funções integradas ao *dashboard* do EasyMQTT, também foi criada uma aplicação simples utilizando o MIT App Inventor[26], uma plataforma também de desenvolvimento em blocos que permite criar

[26] Disponível em: <https://appinventor.mit.edu/>. Acesso em 26 nov. 2020.

aplicativos para smartphone. Na interface da aplicação representada na Figura 35 foram colocados dois botões, um para ligar e ouro para desligar o LED.

Figura 35 – Interface da aplicação desenvolvida no minicurso

Fonte: Cedido pelo Prof. Dr. Rafael V. Aroca.

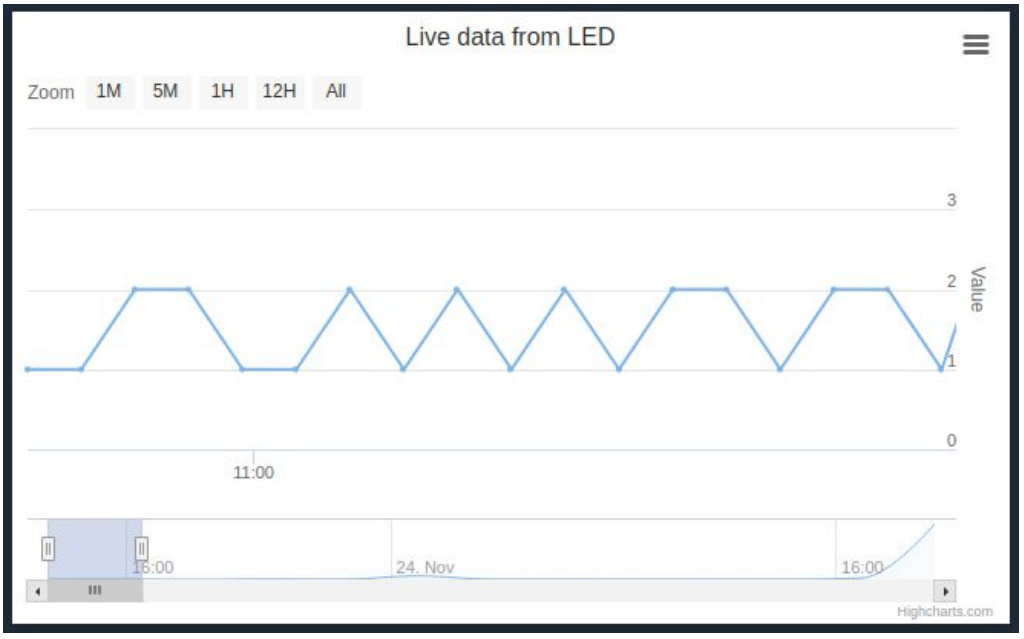
Os cliques em cada um dos botões executa uma chamada para a API de publicação do EasyMQTT, setando o tópico do LED como 1 para desativado ou 2 para ativado. A programação é feita de forma muito simples como é mostrado na Figura 36.

Figura 36 – Programa desenvolvido no MIT App Inventor

Fonte: Cedido pelo Prof. Dr. Rafael V. Aroca.

Abrindo o tópico “1kcyve” na dashboard do EasyMQTT e analisando o gráfico do tópico “LED” (Figura 37) é possível ver que a publicação utilizando o aplicativo desenvolvido funcionou, assumindo os valores requisitados.

Figura 37 – Gráfico do tópico “LED” utilizado no minicurso



Fonte: Elaborado pelo autor.

O EasyMQTT está disponível publicamente no BIPES, portanto tanto os alunos que assistiram ao curso como demais usuários da plataforma já estão fazendo uso de suas funcionalidades. Utilizando a chamada da API do EasyMQTT “listsessions” foram obtidos os dados representados na Tabela 11 no Apêndice F. No momento da escrita deste trabalho o EasyMQTT já contava com 58 sessões diferentes em seu banco de dados, totalizando 87 tópicos e tendo processado o número crescente de 265.431 mensagens.

5 CONCLUSÕES

Este projeto teve como finalidade contribuir para o projeto *open source* BIPES, um projeto recente, mas com muito potencial. A programação sem código provavelmente nunca substituirá completamente as técnicas de programação convencionais ou então terá toda a versatilidade que escrever seu próprio código proporciona, porém abre grandes portas para que mais pessoas adentrem ao mundo da computação e sejam capazes de criarem seus primeiros programas com uma curva de aprendizagem muito menor. Nisto se baseou todo o desenvolvimento efetuado neste trabalho, de criar módulos para comunicação MQTT que fossem além de funcionais, intuitivos e fáceis de se utilizar.

Os testes demonstraram que os módulos estão cumprindo seu papel, validando os objetivos determinados para este trabalho. Também, sabendo que o BIPES é um projeto em constante desenvolvimento e com uma proposta promissora, são listadas abaixo algumas sugestões de melhoria aos módulos MQTT em possíveis trabalhos futuros:

- A implementação de uma forma de isolamento das sessões do EasyMQTT, para que uma aplicação não possa interferir nos dados de outra. Por exemplo, poderia ser desenvolvida uma autenticação por usuário e senha de forma que ainda mantivesse a utilização amigável ao usuário deixando as configurações complicadas sendo realizada automaticamente pela camada de abstração do EasyMQTT;
- A unificação dos serviços do EasyMQTT que atualmente rodam uma parte em Python e outra em PHP todos em uma só linguagem;
- Fazer a portabilidade dos módulos MQTT e EasyMQTT do BIPES para placas que usem o Python comum (como Raspberry Pi) e portanto não utilizam a mesma biblioteca “umqtt”.

REFERÊNCIAS

BIPES. **Welcome to BIPES Project!.** Disponível em: <http://bipes.net.br/>. Acesso em: 1 nov. 2020.

CABALLAR, R. D. **Programming Without Code: The Rise of No-Code Software Development**. IEEE Spectrum Tech Talks. Mar. 2020. Disponível em: <https://spectrum.ieee.org/tech-talk/computing/software/programming-without-code-no-code-software-development>. Acesso em: 1 nov. 2020.

Eletrogate. **NodeMCU v3 Lolin.** Disponível em: <https://www.eletrogate.com/nodemcu-v3-lolin-kit-de-desenvolvimento-com-esp8266-baseado-em-lua>. Acesso em: 25 nov. 2020.

Espressif Systems. **ESP8266 Wi-Fi MCU.** Disponível em: <https://www.espressif.com/en/products/socs/esp8266>. Acesso em: 26 nov. 2020.

HiveMQ. **MQTT Essentials**: The Ultimate Kickstart For MQTT Beginners. 2015. Disponível em: <https://www.hivemq.com/mqtt-essentials/>. Acesso em: 26 nov. 2020.

JUNIOR, A. G. D. S. et al. **BIPES: Block Based Integrated Platform for Embedded Systems**. IEEE Access, v. 8, p. 197955-197968, 2020.

MADEIRA, D. **A Revolução das Placas de Desenvolvimento.** Portal Vida de Silício, 2018. Disponível em: <https://portal.vidadesilicio.com.br/revolucao-das-placas-de-desenvolvimento/>. Acesso em: 26 nov. 2020.

MathWorks. **ThingSpeak Documentation.** Disponível em: <https://www.mathworks.com/help/thingspeak/>. Acesso em 20 nov. 2020.

MicroPython. **Python for Microcontrollers**. Disponível em: <https://micropython.org/>. Acesso em: 25 nov. 2020.

MQTT. **Frequently Asked Questions.** Disponível em: <https://mqtt.org/faq/>. Acesso em: 1 nov. 2020.

POZZEBOM, R. **O que são sistemas embarcados?.** Oficina da Net, 2014. Disponível em: <https://www.oficinadanet.com.br/post/13538-o-que-sao-sistemas-embarcados>. Acesso em: 26 nov. 2020.

Python.org. **About Python.** Disponível em: <https://www.python.org/about/> Acesso em: 01 nov. 2020.

TAN, L., WANG, N. **Future internet: The Internet of Things**. 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), Chengdu, 2010, p. V5-376-V5-380.

WEINTROP, D., WILENSKY, U. **How block-based, text-based, and hybrid**

block/text modalities shape novice programming practices. International Journal of Child-Computer Interaction, v. 17, p. 83-92, 2018.

WEINTROP, D. **Block-based Programming in Computer Science Education.** Communications of the ACM, v. 62, ed. 8, p. 22-25, 2019.

APÊNDICE A – Contribuições à IDE do BIPES

Arquivo “block_definitions.js”

```
/// Pinout
Blockly.Blocks['pinout'] = {
 update_list: function() {
   if (document.getElementById('device_selector').value in pinout){
     this.options = pinout[document.getElementById('device_selector').value];
   }else{
     this.options = [["Pins are not defined","None"]];
   }

 },
 options: [],
 init: function() {
   this.update_list();
   this.appendDummyInput()
       .appendField('pin')
       .appendField(new Blockly.FieldDropdown(this.options), 'PIN');
   this.setOutput(true, null);
   this.setColour(230);
   this.setTooltip("Pins");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// DHT11/22
/// Start DHT Sensor
Blockly.Blocks['dht_init'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldImage(
                   "/beta2/ui/media/dht.png",
                   55,
                   55,
                   "*"))
       .appendField("Start DHT Sensor");
   this.appendDummyInput()
       .appendField('Type')
       .appendField(new Blockly.FieldDropdown([
                   ['DHT11', 'DHT11'],
                   ['DHT22', 'DHT22']
       ]), 'DHT_TYPE');
   this.appendValueInput("pin")
       .setCheck("Number")
       .setAlign(Blockly.ALIGN_RIGHT)
       .appendField(new Blockly.FieldLabelSerializable("pin"), "DHT_PIN_MSG");
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setTooltip("Start DHT11 ou DHT22 sensor");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// Measure DHT11/22 Sensor
Blockly.Blocks['dht_measure'] = {
```

```
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("Measure DHT11/22
Sensor"), "MSG_MEASURE_DHT");
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setTooltip("Measure DHT11/22 Sensor");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// Read DHT11/22 Temperature
Blockly.Blocks['dht_read_temp'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("Read DHT11/22
Temperature"), "MSG_READ_DHT_TEMP");
   this.setOutput(true, null);
   this.setColour(230);
   this.setTooltip("Read DHT11/22 Temperature");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// Read DHT11/22 Humidity
Blockly.Blocks['dht_read_humidity'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("Read DHT11/22 Humidity"),
"MSG_READ_DHT_HUMI");
   this.setOutput(true, null);
   this.setColour(230);
   this.setTooltip("Read DHT11/22 Humidity");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// Relay Switch
Blockly.Blocks['relay_switch'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldImage(
                   "/beta2/ui/media/relay.png",
                   55,
                   55,
                   "*"))
       .setAlign(Blockly.ALIGN_CENTRE)
       .appendField("Relay");
   this.appendValueInput("pin")
       .setCheck("Number")
       .appendField('Turn')
       .appendField(new Blockly.FieldDropdown([
                     ['off', '0'],
                     ['on', '1']
                   ]), 'RELAY_STATUS')
       .appendField('relay on pin');
   this.setPreviousStatement(true, null);
```

```
    this.setNextStatement(true, null);
    this.setColour(230);
    this.setTooltip("Turn On Relay on GPIO digital pin");
    this.setHelpUrl("bipes.net.br");
  }
};

/// MQTT
/// Start MQTT Client
Blockly.Blocks['mqtt_init'] = {
  init: function() {
    this.appendDummyInput()
        .appendField(new Blockly.FieldLabelSerializable("Start MQTT Client"),
"BLOCK_MQTT_INIT");
    this.appendValueInput("server")
        .setCheck("String")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldLabelSerializable("Server Address"),
"MQTT_SERVER");
    this.appendValueInput("port")
        .setCheck("Number")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldLabelSerializable("Server Port"),
"MQTT_PORT");
    this.appendValueInput("user")
        .setCheck("String")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldLabelSerializable("Username"), "MQTT_USER");
    this.appendValueInput("password")
        .setCheck("String")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldLabelSerializable("Password"),
"MQTT_PASSWORD");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
    this.setTooltip("Start MQTT Client");
    this.setHelpUrl("http://www.bipes.net.br");
  }
};

/// Add Data to MQTT Buffer
Blockly.Blocks['mqtt_add_to_buffer'] = {
  init: function() {
    this.appendDummyInput()
        .appendField(new Blockly.FieldLabelSerializable("Add Data to MQTT
Buffer"), "BLOCK_MQTT_ADD_TO_BUFFER");
    this.appendValueInput("fieldname")
        .setCheck("String")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldLabelSerializable("Field Name"),
"MQTT_FIELDNAME");
    this.appendValueInput("value")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldLabelSerializable("Value"), "MQTT_VALUE");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
```

```
    this.setTooltip("Add Data to MQTT Buffer");
    this.setHelpUrl("http://www.bipes.net.br");
  }
};

/// Publish Buffer to MQTT Topic
Blockly.Blocks['mqtt_publish_buffer'] = {
  init: function() {
    this.appendDummyInput()
        .appendField(new Blockly.FieldLabelSerializable("Publish Buffer to MQTT
Topic"), "BLOCK_MQTT_PUBLISH");
    this.appendValueInput("topic")
        .setCheck("String")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldLabelSerializable("Topic"), "MQTT_TOPIC");
    this.appendDummyInput()
        .appendField('QOS:')
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldDropdown([
            ['0 - at most\u00A0once', '0'],
            ['1 - at least\u00A0once', '1']
        ]), 'MQTT_QOS');
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
    this.setInputsInline(false);
    this.setTooltip("Publish Buffer to MQTT Server");
    this.setHelpUrl("http://www.bipes.net.br");
  }
};

/// Publish Payload to MQTT Topic
Blockly.Blocks['mqtt_publish_payload'] = {
  init: function() {
    this.appendDummyInput()
        .appendField(new Blockly.FieldLabelSerializable("Publish Payload to MQTT
Topic"), "BLOCK_MQTT_PUBLISH");
    this.appendValueInput("topic")
        .setCheck("String")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldLabelSerializable("Topic"), "MQTT_TOPIC");
    this.appendValueInput("payload")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldLabelSerializable("Payload"),
"MQTT_PAYLOAD");
    this.appendDummyInput()
        .appendField('QOS:')
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField(new Blockly.FieldDropdown([
            ['0 - at most\u00A0once', '0'],
            ['1 - at least\u00A0once', '1']
        ]), 'MQTT_QOS');
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
    this.setInputsInline(false);
    this.setTooltip("Publish Payload to MQTT Server");
    this.setHelpUrl("http://www.bipes.net.br");
```

```
 }
};

/// Subscribe to MQTT Topic
Blockly.Blocks['mqtt_subscribe'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("Subscribe to MQTT
Topic"), "BLOCK_MQTT_SUBSCRIBE");
   this.appendValueInput("topic")
       .setCheck("String")
       .setAlign(Blockly.ALIGN_RIGHT)
       .appendField(new Blockly.FieldLabelSerializable("Topic"), "MQTT_TOPIC");
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setTooltip("Subscribe to MQTT Topic");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// Set Callback to MQTT Messages
Blockly.Blocks['mqtt_set_callback'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("Set Callback to MQTT
Messages"), "BLOCK_MQTT_SET_CALLBACK");
   this.appendDummyInput()
       .appendField('with')
       .appendField(new Blockly.FieldVariable('data_bytes'), 'MQTT_DATA_VAR')
       .appendField('received from')
       .appendField(new Blockly.FieldVariable(
         'topic',
         null,
         ['String'],
         'String'
       ), 'MQTT_TOPIC_VAR');
   this.appendStatementInput('do')
       .appendField('do');
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setInputsInline(false);
   this.setTooltip("Callback function must have topic and msg parameters");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// Check MQTT Server for pending messages
Blockly.Blocks['mqtt_check_msg'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("Check MQTT Server for
pending messages"), "BLOCK_MQTT_CHECK_MSG");
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setTooltip("Check if the server has any pending messages. Non-blocking
```

```
method. Subscription messages will be passed to the callback.");
    this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// Wait for MQTT Server messages
Blockly.Blocks['mqtt_wait_msg'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("Wait for MQTT Server
messages"), "BLOCK_MQTT_WAIT_MSG");
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setTooltip("Wait for server sending any message. Blocking method.
Subscription messages will be passed to the callback.");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// Disconnect MQTT Client
Blockly.Blocks['mqtt_disconnect'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("Disconnect MQTT Client"),
"BLOCK_MQTT_DISCONNECT");
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setTooltip("Disconnect the MQTT Client from Server.");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// Convert to Str
Blockly.Blocks['text_to_str'] = {
 init: function() {
   this.appendValueInput("var")
       .appendField(new Blockly.FieldLabelSerializable("to str"), "VAR");
   this.setColour(160);
   this.setOutput(true, null);
   this.setTooltip("Convert anything to String.");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// Decode Bytes to Str
Blockly.Blocks['decode_bytes_to_text'] = {
 init: function() {
   this.appendValueInput("var")
       .appendField(new Blockly.FieldLabelSerializable("decode bytes to text"),
"VAR");
   this.setColour(160);
   this.setOutput(true, null);
   this.setTooltip("Decode bytes to a String.");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};
```

```
/// Convert to Int
Blockly.Blocks['var_to_int'] = {
 init: function() {
   this.appendValueInput("var")
       .appendField(new Blockly.FieldLabelSerializable("to int"), "VAR");
   this.setColour(230);
   this.setOutput(true, null);
   this.setTooltip("Convert anything to Int.");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// Convert to Float
Blockly.Blocks['var_to_float'] = {
 init: function() {
   this.appendValueInput("var")
       .appendField(new Blockly.FieldLabelSerializable("to float"), "VAR");
   this.setColour(230);
   this.setOutput(true, null);
   this.setTooltip("Convert anything to float.");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// EasyMQTT
/// EasyMQTT Init
Blockly.Blocks['easymqtt_init'] = {
 generate_id: function(){
   return Math.random().toString(36).substring(7);
 },
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("EasyMQTT Start"),
"BLOCK_EASYMQTT_INIT");
   this.appendDummyInput()
       .appendField("Session ID")
       .appendField(new Blockly.FieldTextInput(this.generate_id()),
           'EASYMQTT_SESSION_ID');
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setTooltip("Start EasyMQTT Client");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// EasyMQTT Publish Data
Blockly.Blocks['easymqtt_publish_data'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("EasyMQTT Publish Data"),
"BLOCK_EASYMQTT_PUBLISH");
   this.appendValueInput("topic")
       .setCheck("String")
       .setAlign(Blockly.ALIGN_RIGHT)
       .appendField(new Blockly.FieldLabelSerializable("Topic"),
"EASYMQTT_TOPIC");
```

```
   this.appendValueInput("data")
       .setAlign(Blockly.ALIGN_RIGHT)
       .setCheck("Number")
       .appendField(new Blockly.FieldLabelSerializable("Data"),
"EASYMQTT_PAYLOAD");
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setTooltip("Publish Data to EasyMQTT Server");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

///EasyMQTT Subscribe
Blockly.Blocks['easymqtt_subscribe'] = {
 init: function() {
   this.appendValueInput("topic")
       .setCheck("String")
       .setAlign(Blockly.ALIGN_RIGHT)
       .appendField(new Blockly.FieldLabelSerializable("EasyMQTT Subscribe to
Topic"), "EASYMQTT_TOPIC");
   this.appendDummyInput()
       .appendField('when')
       .appendField(new Blockly.FieldVariable(
         'data',
         null,
         ['Number'],
         'Number'
       ), 'EASYMQTT_VAR')
       .appendField('is received');
   this.appendStatementInput('do')
       .appendField('do');
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setInputsInline(false);
   this.setTooltip("Subscribe to a topic and define what to do when data is
received from EasyMQTT Server");
   this.setHelpUrl("http://www.bipes.net.br");
 }
};

/// EasyMQTT Receive Data
Blockly.Blocks['easymqtt_receive_data'] = {
 init: function() {
   this.appendDummyInput()
       .appendField(new Blockly.FieldLabelSerializable("EasyMQTT Receive Data"),
"BLOCK_EASYMQTT_RECEIVE");
   this.appendDummyInput()
       .appendField('wait for data')
       .appendField(new Blockly.FieldDropdown([
           ['no', '0'],
           ['yes', '1']
       ]), 'EASYMQTT_WAIT');
   this.setPreviousStatement(true, null);
   this.setNextStatement(true, null);
   this.setColour(230);
   this.setTooltip("Receive Data from EasyMQTT Server");
```

```
    this.setHelpUrl("http://www.bipes.net.br");
  }
};

/// EasyMQTT Disconnect
Blockly.Blocks['easymqtt_disconnect'] = {
  init: function() {
    this.appendDummyInput()
        .appendField(new Blockly.FieldLabelSerializable("EasyMQTT Stop"),
"BLOCK_EASYMQTT_DISCONNECT");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
    this.setTooltip("Disconnect the EasyMQTT Client from Server.");
    this.setHelpUrl("http://www.bipes.net.br");
  }
};
```

Arquivo “generator_stubs.js”

```
/// Pinout
Blockly.Python['pinout'] = function(block) {
  var pin = block.getFieldValue('PIN');
   return [pin, Blockly.Python.ORDER_NONE];
};

/// DHT11/22
/// Start DHT Sensor
Blockly.Python['dht_init'] = function(block) {
  var value_pin = Blockly.Python.valueToCode(block, 'pin',
Blockly.Python.ORDER_ATOMIC);
  var type = block.getFieldValue('DHT_TYPE');
  Blockly.Python.definitions_['import_machine'] = 'import machine';
  Blockly.Python.definitions_['import_dht'] = 'import dht';
  Blockly.Python.definitions_['import_time'] = 'import time';
  var code = 'dhts=dht.' + type + '(machine.Pin(' + value_pin +
'));dhts.measure();time.sleep(2)\n';
  return code;
};

/// Measure DHT11/22 Sensor
Blockly.Python['dht_measure'] = function(block) {
  var code = 'dhts.measure()\n';
  return code;
};

/// Read DHT11/22 Temperature
Blockly.Python['dht_read_temp'] = function(block) {
  var code = 'dhts.temperature()';
  return [code, Blockly.Python.ORDER_NONE];
};

/// Read DHT11/22 Humidity
Blockly.Python['dht_read_humidity'] = function(block) {
  var code = 'dhts.humidity()';
  return [code, Blockly.Python.ORDER_NONE];
};
```

```
/// Relay Switch
Blockly.Python['relay_switch'] = function(block) {
 var pin = Blockly.Python.valueToCode(block, 'pin', Blockly.Python.ORDER_ATOMIC);
 var status = block.getFieldValue('RELAY_STATUS');
 Blockly.Python.definitions_['import_machine'] = 'import machine';
 if (status == '1'){
   var code = 'machine.Pin(' + pin + ').off()\n';
 }else{
   var code = 'machine.Pin(' + pin + ').on()\n';
 }
 return code;
};

/// MQTT
/// Start MQTT Client
Blockly.Python['mqtt_init'] = function(block) {
 var server = Blockly.Python.valueToCode(block, 'server',
Blockly.Python.ORDER_ATOMIC);
 var port = Blockly.Python.valueToCode(block, 'port',
Blockly.Python.ORDER_ATOMIC);
 var user = Blockly.Python.valueToCode(block, 'user',
Blockly.Python.ORDER_ATOMIC);
 var pass = Blockly.Python.valueToCode(block, 'password',
Blockly.Python.ORDER_ATOMIC);

 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';
 var code = 'mqtt_buffer = ""; mqtt_client =
umqtt.robust.MQTTClient("umqtt_client", server = ' + server + ', port = ' + port
+ ', user = ' + user + ', password = ' + pass + '); mqtt_client.connect()\n'
 return code;
};

/// Add Data to MQTT Buffer
Blockly.Python['mqtt_add_to_buffer'] = function(block) {
 var name = Blockly.Python.valueToCode(block, 'fieldname',
Blockly.Python.ORDER_ATOMIC);
 var value = Blockly.Python.valueToCode(block, 'value',
Blockly.Python.ORDER_ATOMIC);

 var code = 'mqtt_buffer += (' + name + ' + "=" + str(' + value + ')) if not
len(mqtt_buffer) else ("&" + ' + name + ' + "=" + str(' + value + '))\n'
 return code;
};

/// Publish Buffer to MQTT Topic
Blockly.Python['mqtt_publish_buffer'] = function(block) {
 var topic = Blockly.Python.valueToCode(block, 'topic',
Blockly.Python.ORDER_ATOMIC);
 var qos = block.getFieldValue('MQTT_QOS');

 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';

 var code = 'mqtt_client.publish(' + topic + ', mqtt_buffer,qos=' + qos + ');
mqtt_buffer = ""\n';
 return code;
};
```

```
/// Publish Payload to MQTT Topic
Blockly.Python['mqtt_publish_payload'] = function(block) {
 var topic = Blockly.Python.valueToCode(block, 'topic',
Blockly.Python.ORDER_ATOMIC);
 var payload = Blockly.Python.valueToCode(block, 'payload',
Blockly.Python.ORDER_ATOMIC);
 var qos = block.getFieldValue('MQTT_QOS');

 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';

 var code = 'mqtt_client.publish(' + topic + ', ' + payload + ',qos=' + qos +
')\n';
 return code;
};

/// Set Callback to MQTT Messages
Blockly.Python['mqtt_set_callback'] = function(block) {
   var data_var_name =
Blockly.Python.variableDB_.getName(block.getFieldValue('MQTT_DATA_VAR'),
Blockly.VARIABLE_CATEGORY_NAME);
   var topic_var_name =
Blockly.Python.variableDB_.getName(block.getFieldValue('MQTT_TOPIC_VAR'),
Blockly.VARIABLE_CATEGORY_NAME);
   // Fix for global variables inside callback
   // Piece of code from generators/python/procedures.js
   // Add a 'global' statement for every variable that is not shadowed by a local
parameter.
   var globals = [];
   var varName;
   var workspace = block.workspace;
   var variables = Blockly.Variables.allUsedVarModels(workspace) || [];
   for (var i = 0, variable; variable = variables[i]; i++) {
       varName = variable.name;
       if (block.getVars().indexOf(varName) == -1 && varName != data_var_name &&
varName != topic_var_name) {
       globals.push(Blockly.Python.variableDB_.getName(varName,
           Blockly.VARIABLE_CATEGORY_NAME));
       }
   }
   // Add developer variables.
   var devVarList = Blockly.Variables.allDeveloperVariables(workspace);
   for (var i = 0; i < devVarList.length; i++) {
       globals.push(Blockly.Python.variableDB_.getName(devVarList[i],
           Blockly.Names.DEVELOPER_VARIABLE_TYPE));
   }
   globals = globals.length ? Blockly.Python.INDENT + 'global ' + globals.join(',
') : '';
   // End of code from generators/python/procedures.js

   Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';

   var funct_code = Blockly.Python.statementToCode(block, 'do');


   var function_name = Blockly.Python.provideFunction_(
       'mqtt_callback',
       ['def ' + Blockly.Python.FUNCTION_NAME_PLACEHOLDER_ +
'('+topic_var_name+','+data_var_name+'):',
```

```
       globals,
       Blockly.Python.INDENT + topic_var_name + " = " + topic_var_name +
".decode()",
       funct_code]);

   var code = 'mqtt_client.set_callback(' + function_name + ')\n';
   return code;
};

/// Subscribe to MQTT Topic
Blockly.Python['mqtt_subscribe'] = function(block) {
 var topic = Blockly.Python.valueToCode(block, 'topic',
Blockly.Python.ORDER_ATOMIC);

 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';

 var code = 'mqtt_client.subscribe(' + topic + ')\n';
 return code;
};

/// Check for MQTT Server messages
Blockly.Python['mqtt_check_msg'] = function(block) {
 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';

 var code = 'mqtt_client.check_msg()\n';
 return code;
};

/// Wait for MQTT Server messages
Blockly.Python['mqtt_wait_msg'] = function(block) {
 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';

 var code = 'mqtt_client.wait_msg()\n';
 return code;
};

/// Disconnect MQTT Client
Blockly.Python['mqtt_disconnect'] = function(block) {
 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';

 var code = 'mqtt_client.disconnect()\n';
 return code;
};

/// Convert to Str
Blockly.Python['text_to_str'] = function(block) {
 var variable = Blockly.Python.valueToCode(block, 'var',
Blockly.Python.ORDER_ATOMIC);
 var code = 'str(' + variable + ')';

 return [code, Blockly.Python.ORDER_NONE];
};

/// Decode Bytes to Str
Blockly.Python['decode_bytes_to_text'] = function(block) {
 var variable = Blockly.Python.valueToCode(block, 'var',
Blockly.Python.ORDER_ATOMIC);
 var code =  variable + '.decode()';
```

```
 return [code, Blockly.Python.ORDER_NONE];
};

/// Convert to Int
Blockly.Python['var_to_int'] = function(block) {
 var variable = Blockly.Python.valueToCode(block, 'var',
Blockly.Python.ORDER_ATOMIC);
 var code = 'int(' + variable + ')';

 return [code, Blockly.Python.ORDER_NONE];
};
/// Convert to Float
Blockly.Python['var_to_float'] = function(block) {
 var variable = Blockly.Python.valueToCode(block, 'var',
Blockly.Python.ORDER_ATOMIC);
 var code = 'float(' + variable + ')';
  return [code, Blockly.Python.ORDER_NONE];
};

/// EasyMQTT
/// EasyMQTT Init
Blockly.Python['easymqtt_init'] = function(block) {
 var server = '"bipes.net.br"';
 var port = '1883';
 var user = '"bipes"';
 var pass = '"m8YLUr5uW3T"';
 var session = block.getFieldValue('EASYMQTT_SESSION_ID');
 window.easyMQTT_session = session;

 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';
 var code = 'easymqtt_session = "' + session + '"; easymqtt_client =
umqtt.robust.MQTTClient("umqtt_client", server = ' + server + ', port = ' + port
+ ', user = ' + user + ', password = ' + pass + ');
easymqtt_client.connect()\nprint("EasyMQTT connected")\n'
 return code;
};

/// EasyMQTT Publish Data
Blockly.Python['easymqtt_publish_data'] = function(block) {
 var topic = Blockly.Python.valueToCode(block, 'topic',
Blockly.Python.ORDER_ATOMIC);
 var data = Blockly.Python.valueToCode(block, 'data',
Blockly.Python.ORDER_ATOMIC);

 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';

 var code = 'easymqtt_client.publish(easymqtt_session + "/" + ' + topic + ',
str(' + data + '))\nprint("EasyMQTT Publish -
Session:",easymqtt_session,"Topic:",' + topic + ',"Value:",str(' + data + '))\n'
 return code;
};

/// EasyMQTT Disconnect
Blockly.Python['easymqtt_disconnect'] = function(block) {
 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';

 var code = 'easymqtt_client.disconnect()\nprint("EasyMQTT disconnected")\n';
```

```
 return code;
};

///EasyMQTT Subscribe
Blockly.Python['easymqtt_subscribe'] = function(block) {
 var var_name = Blockly.Python.variableDB_.getName(
     block.getFieldValue('EASYMQTT_VAR'), Blockly.VARIABLE_CATEGORY_NAME);
 // Fix for global variables inside callback
 // Piece of code from generators/python/procedures.js
 // Define a procedure with a return value.
 // First, add a 'global' statement for every variable that is not shadowed by
 // a local parameter.
 var globals = [];
 var varName;
 var workspace = block.workspace;
 var variables = Blockly.Variables.allUsedVarModels(workspace) || [];
 for (var i = 0, variable; variable = variables[i]; i++) {
   varName = variable.name;
   if (block.getVars().indexOf(varName) == -1 && varName != var_name) {
     globals.push(Blockly.Python.variableDB_.getName(varName,
         Blockly.VARIABLE_CATEGORY_NAME));
   }
 }
 // Add developer variables.
 var devVarList = Blockly.Variables.allDeveloperVariables(workspace);
 for (var i = 0; i < devVarList.length; i++) {
   globals.push(Blockly.Python.variableDB_.getName(devVarList[i],
       Blockly.Names.DEVELOPER_VARIABLE_TYPE));
 }
 globals = globals.length ? Blockly.Python.INDENT + 'global ' + globals.join(',
') + '\n' : '';

 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';
 var topic = Blockly.Python.valueToCode(block, 'topic',
Blockly.Python.ORDER_ATOMIC);
 var funct_code = Blockly.Python.statementToCode(block, 'do');
 var name = topic.replace(/\W/g, '_');

 var function_name = Blockly.Python.provideFunction_(
   'easymqtt'+name,
   ['def ' + Blockly.Python.FUNCTION_NAME_PLACEHOLDER_ +
'('+var_name+'):',globals,funct_code]);

 Blockly.Python.definitions_['easymqtt_callback'] = 'easymqtt_callback_list =
{}\ndef easymqtt_callback(topic_,msg_):\n
topic_=topic_.decode();msg_=msg_.decode()\n  if topic_ in easymqtt_callback_list:
easymqtt_callback_list[topic_](float(msg_))';

 var code =
"easymqtt_client.set_callback(easymqtt_callback)\neasymqtt_callback_list['"+windo
w.easyMQTT_session+"/' +
"+topic+"]="+function_name+"\neasymqtt_client.subscribe('"+window.easyMQTT_sessio
n+"/' + "+topic+")\n"
 return code;
};

/// EasyMQTT Receive Data
Blockly.Python['easymqtt_receive_data'] = function(block) {
```

```
 Blockly.Python.definitions_['import_umqtt.robust'] = 'import umqtt.robust';
 var wait = block.getFieldValue('EASYMQTT_WAIT');
 if (wait == '1'){
   var code = 'easymqtt_client.wait_msg()\n';
 }else{
   var code = 'easymqtt_client.check_msg()\n';
 }
 return code;
};
```

```
<td id="tab_mqtt" class="taboff"
onclick='document.getElementById("easymqtt_iframe").contentWindow.startEasyMQTT(w
indow.easyMQTT_session);'>EasyMQTT</td>
<td class="tabmin"> </td>
[...]
var pinout = {
   'ESP8266': [
     ['D0', "16"],
     ['D1', "5"],
     ['D2', "4"],
     ['D3', "0"],
     ['D4/LED', "2"],
     ['D5', "14"],
     ['D6', "12"],
     ['D7', "13"],
     ['D8', "15"],
     ['RX', "3"],
     ['TX', "1"],
     ['SD3', "10"],
     ['SD2', "9"],
     ['SD1', "8"],
     ['CMD', "11"],
     ['SD0', "7"],
     ['CLK', "6"]
   ]
 };
[...]
<div id="content_mqtt" class="content" style="padding: 0;">
   <iframe id="easymqtt_iframe" src="../easymqtt/index.html" width="100%"
height="100%" frameBorder="0"></iframe>
</div>
[...]
<category name="MQTT">
   <label text="MQTT (Message Queue Telemetry Transport)"></label>
   <block type="mqtt_init">
     <field name="BLOCK_MQTT_INIT">Start MQTT Client</field>
     <value name="server">
       <shadow type="text">
         <field name="TEXT"></field>
       </shadow>
     </value>
     <value name="port">
       <shadow type="math_number">
         <field name="NUM">1883</field>
```

```
      </shadow>
    </value>
    <value name="user">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
    <value name="password">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
  </block>
  <block type="mqtt_add_to_buffer">
    <field name="BLOCK_MQTT_ADD_TO_BUFFER">Add data to MQTT Buffer</field>
    <value name="fieldname">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
    <field name="MQTT_VALUE">Value</field>
  </block>
  <block type="mqtt_publish_buffer">
    <field name="BLOCK_MQTT_PUBLISH">Publish Buffer to MQTT Topic</field>
    <value name="topic">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
  </block>
  <block type="mqtt_publish_payload">
    <field name="BLOCK_MQTT_PUBLISH">Publish Payload to MQTT Topic</field>
    <value name="topic">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
    <value name="payload">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
  </block>
  <block type="mqtt_set_callback">
    <field name="BLOCK_MQTT_SET_CALLBACK">Set Callback to MQTT Messages</field>
  </block>
  <block type="mqtt_subscribe">
    <field name="BLOCK_MQTT_SUBSCRIBE">Subscribe to MQTT Topic</field>
    <value name="topic">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
  </block>
  <block type="mqtt_check_msg">
    <field name="BLOCK_MQTT_CHECK_MSG">Check for MQTT Server messages</field>
  </block>
  <block type="mqtt_wait_msg">
```

```
    <field name="BLOCK_MQTT_WAIT_MSG">Wait for MQTT Server messages</field>
  </block>
  <block type="mqtt_disconnect">
    <field name="BLOCK_MQTT_DISCONNECT">Disconnect MQTT Client</field>
  </block>
 </category>
 <category name="EasyMQTT">
  <label text="IoT: EasyMQTT"></label>
  <block type="easymqtt_init">
    <field name="BLOCK_EASYMQTT_INIT">EasyMQTT Start</field>
  </block>
  <block type="easymqtt_publish_data">
    <field name="BLOCK_EASYMQTT_PUBLISH">EasyMQTT Publish Data</field>
    <value name="topic">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
    <value name="data">
      <shadow type="math_number">
        <field name="NUM"></field>
      </shadow>
    </value>
  </block>
  <block type="easymqtt_subscribe">
    <value name="topic">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
  </block>
  <block type="easymqtt_receive_data"></block>
  <block type="easymqtt_disconnect">
    <field name="BLOCK_EASYMQTT_DISCONNECT">EasyMQTT Stop</field>
  </block>
 </category>
[...]
 <category name="Relay">
  <label text="Relay"></label>
  <block type="relay_switch">
    <value name="pin">
      <shadow type="pinout">
        <field name="PIN"></field>
      </shadow>
    </value>
  </block>
 </category>
[...]
 <category name="DHT11/22 Sensor">
  <label text="DHT11/22 Temperature and Humidity Sensor"></label>
  <block type="dht_init">
    <field name="BLOCK_DHT_INIT">Start DHT11/22 sensor</field>
    <value name="pin">
      <shadow type="pinout">
        <field name="PIN"></field>
      </shadow>
    </value>
  </block>
```

```
    <block type="dht_measure">
      <field name="MSG_MEASURE_DHT">Measure DHT11/22 Sensor</field>
    </block>
    <block type="dht_read_temp">
      <field name="MSG_READ_DHT_TEMP">Read DHT11/22 Temperature</field>
    </block>
    <block type="dht_read_humidity">
      <field name="MSG_READ_DHT_HUMI">Read DHT11/22 Humidity</field>
    </block>
  </category>
[...]
  <block type="pinout"></block>
[...]
  <block type="var_to_int">
    <field name="VAR">to int</field>
  </block>
  <block type="var_to_float">
    <field name="VAR">to float</field>
  </block>
[...]
  <block type="text_to_str">
    <field name="VAR">to str</field>
  </block>
  <block type="decode_bytes_to_text">
    <field name="VAR">decode bytes to text</field>
  </block>
```

APÊNDICE B – Códigos fonte da API do EasyMQTT

Arquivo “listsessions.php”

```
<?php
header('Content-Type: application/json');
header("Access-Control-Allow-Origin: *");

require("vendor/autoload.php");

$client = new MongoDB\Client("mongodb://localhost:27017");

$result= array();
foreach ($client->listDatabaseNames() as $databaseName) {
   $db = $client->selectDatabase($databaseName);
   $totalMessages = 0;
   $totalTopics = 0;
   foreach ($db->listCollectionNames() as $collectionName) {
       $totalTopics += 1;
       $collection = $db->selectCollection($collectionName);
       $totalMessages += $collection->count();
   }
   $result[] = array("session" => $databaseName, "topicsCount" => $totalTopics,
"messagesCount" => $totalMessages);
}

$return = array("success" => True, "result" => $result);

echo(json_encode($return));
?>
```

Arquivo “getsession.php”

```
<?php
header('Content-Type: application/json');
header("Access-Control-Allow-Origin: *");
if(!isset($_GET['session']) || empty($_GET['session']))
{
   echo(json_encode(array("success" => False, "result" => "Invalid
Parameters")));
   die();
}
require("vendor/autoload.php");

$client = new MongoDB\Client("mongodb://localhost:27017");
$session = htmlspecialchars($_GET["session"]);

$db = $client->selectDatabase($session);
$results = $db->listCollections();

$topics = array();
foreach ($results as $collection) {
   $topics[] = $collection['name'];
}
if (count($topics)>0)
   $return = array("success" => True, "result" => $topics);
else
```

```
   $return = array("success" => False, "result" => "Session '" . $session . "' is
empty.<br /><span>As soon as something reaches the broker, it will be displayed
here!</span>");

echo(json_encode($return));
?>
```

Arquivo “gettopic.php”

```
<?php
header('Content-Type: application/json');
header("Access-Control-Allow-Origin: *");
if(!isset($_GET['session']) || empty($_GET['session']) || !isset($_GET['topic'])
|| empty($_GET['topic']))
{
   echo(json_encode(array("success" => False, "result" => "Invalid
Parameters")));
   die();
}

require("vendor/autoload.php");

$client = new MongoDB\Client("mongodb://localhost:27017");
$session = htmlspecialchars($_GET["session"]);
$topic = htmlspecialchars($_GET["topic"]);

$db = $client->selectDatabase($session);
$collection = $db->selectCollection($topic);

if (isset($_GET["since"]) && !empty($_GET["since"])){
   $since = htmlspecialchars($_GET["since"]);
   $results = $collection->find(['timestamp' => ['$gte' => intval($since)]]);
}else{
   $results = $collection->find();
}

$values = array();
foreach ($results as $item) {
   $values[] = array("timestamp" => $item['timestamp'], "data" => $item['data']);
}
$return = array("success" => True, "result" => $values);

echo(json_encode($return));
?>
```

Arquivo “publish.php”

```
<?php
header('Content-Type: application/json');
header("Access-Control-Allow-Origin: *");
if(!isset($_GET['session']) || empty($_GET['session']) || !isset($_GET['topic'])
|| empty($_GET['topic']) || !isset($_GET['value']))
{
   echo(json_encode(array("success" => False, "result" => "Invalid
Parameters")));
   die();
}
```

```
$session = htmlspecialchars($_GET["session"]);
$topic = htmlspecialchars($_GET["topic"]);
$value = htmlspecialchars($_GET["value"]);

if (!is_numeric($value)){
   $return = array("success" => False, "result" => "Error publishing value
'".$value."' to topic '".$topic."'. Non-numeric input value!");

}else{
   $ret;
   $out;

   $err=exec("python3 server/publish.py ".$session."/".$topic."
".$value,$out,$ret);
   if ($ret == 0)
       $return = array("success" => True, "result" => "Value '".$value."'
published to topic '".$topic."' successfully!");
   else
       $return = array("success" => False, "result" => "Error publishing value
'".$value."' to topic '".$topic."'. ".$err);
}

echo(json_encode($return));
?>
```

Arquivo “clearsession.php”

```
<?php
header('Content-Type: application/json');
header("Access-Control-Allow-Origin: *");
if(!isset($_GET['session']) || empty($_GET['session']))
{
   echo(json_encode(array("success" => False, "result" => "Invalid
Parameters")));
   die();
}
require("vendor/autoload.php");

$client = new MongoDB\Client("mongodb://localhost:27017");
$session = htmlspecialchars($_GET["session"]);

$client->dropDatabase($session);
$return = array("success" => True, "result" => "Session '" . $session . "'
cleaned");

echo(json_encode($return));

?>
```

APÊNDICE C – Códigos fonte da Dashboard do EasyMQTT

Arquivo “index.html”

```
<!DOCTYPE HTML>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <meta name="author" content="Caio Augusto Silva - BIPES Project">
 <meta name="description" content="EasyMQTT dashboard for data visualization">
 <title>BIPES EasyMQTT</title>
 <style>
 body{
   background-color: #1f2b38;
   font-family: sans-serif;
   margin:0;
 }

 .easymqtt-content{
   padding: 1ex;
 }

 .card, .highcharts-data-table{
   margin: 10px;
   box-shadow: 0 4px 8px 0 rgba(0,0,0,0.2);
   background-color: #fff;
 }

 .card:hover, .highcharts-data-table:hover{
   box-shadow: 0 8px 16px 0 rgba(0,0,0,0.2);
   transition: 0.3s;
 }

 .highcharts-data-table{
   max-height: 400px;
   overflow-y: scroll;
 }

 .easymqtt-charts{
   display: flex;
   flex-direction: column;
   flex-wrap: wrap;
   justify-content: center;
 }
 @media only screen and (min-width: 768px) {
   .easymqtt-charts{
     flex-direction: row;
   }
 }

 .easymqtt-config{
   display: flex;
   flex-direction: row;
   font-size:14px;
   flex-wrap: wrap;
   justify-content: space-evenly;
 }
```

```
 .easymqtt-config button{
   font-size:14px;
 }

 .easymqtt-config div{
   margin: 10px;
 }

 .easymqtt-chart{
   flex: 1;
   min-width: 280px;
 }

 @media only screen and (min-width: 768px) {
   .easymqtt-chart{
     min-width: 500px;
   }
 }
 .easymqtt-warning{
   text-align: center;
   font-size: 18px;
   font-weight: bold;
   padding: 20px;
 }
 .easymqtt-warning span{
   font-size: 12px;
   font-weight: 400;
 }

 </style>
 <script src="jquery-3.5.1.min.js"></script>
 <script src="https://code.highcharts.com/stock/highstock.js"></script>
 <script src="https://code.highcharts.com/stock/modules/exporting.js"></script>
 <script src="https://code.highcharts.com/stock/modules/export-data.js"></script>
</head>
<body>
 <div id="easymqtt-content" class="content easymqtt-content">
   <div id="easymqtt-warning" class="easymqtt-warning card"></div>
   <div id="easymqtt-charts" class="easymqtt-charts"></div>
   <div id="easymqtt-config" class="easymqtt-config card" style="display: none;">
     <div id="easymqtt-config-publish">
       <label for="easymqtt-publish-value">Publish value:</label>
       <input type="number" id="easymqtt-publish-value"
name="easymqtt-publish-value" step="0.1" value="0" style="width: 40px;">
       <label for="easymqtt-publish-topic">to topic:</label>
       <input type="text" id="easymqtt-publish-topic"
name="easymqtt-publish-topic" style="width: 100px;">
       <button id="easymqtt-publish-submit" type="button"
onclick="publishEasyMQTT($('#easymqtt-publish-topic').val(),$('#easymqtt-publish-
value').val())">Submit</button>
     </div>
     <div id="easymqtt-config-update">
       <label for="easymqtt-update-freq">Update frequency (s):</label>
       <input type="number" id="easymqtt-update-freq" name="easymqtt-update-freq"
min="1" step="1" value="5" style="width: 40px;">
       <button id="easymqtt-update-freq-set" type="button"
onclick="setEasyMQTTUpdate($('#easymqtt-update-freq').val())">Set</button>
```

```
      </div>
      <div id="easymqtt-config-clear">
        <button id="easymqtt-clear-data" type="button"
onclick="clearEasyMQTTSession()">Clear Data</button>
      </div>
    </div>
  </div>
  <script type="text/javascript">
    var timerUpdateEasyMQTT;
    var MQTTCharts = {};
    var easyMQTT_session_chart;

    function getSessionFromURL(){
      let searchParams = new URLSearchParams(window.location.search);
      if (searchParams.has('session')){
        return searchParams.get('session');
      }
      return null;
    }

    // Init
    let urlSession = getSessionFromURL();
    if (urlSession){
      startEasyMQTT(urlSession);
    }else{
      if (parent === window){
        errorEasyMQTT("EasyMQTT Session ID is not currently defined!<br><span>Try
passing it as parameter in URL (?session=<session_id>)</span>");
      }else{
          errorEasyMQTT("EasyMQTT Session ID is not currently defined!<br><span>If
the EasyMQTT Start block is set try clicking on 'Python' tab first.</span>");
      }
    }

    function startEasyMQTT(session){
      if (typeof session === 'undefined'){
        errorEasyMQTT("EasyMQTT Session ID is not currently defined!<br><span>If
the EasyMQTT Start block is set try clicking on 'Python' tab first.</span>");
        if (typeof timerUpdateEasyMQTT !== 'undefined')
          cancelTimer(timerUpdateEasyMQTT);
      }else{
        if (easyMQTT_session_chart == session && typeof timerUpdateEasyMQTT !==
'undefined'){
          return;
        }
        if (typeof easyMQTT_session_chart !== 'undefined' &&
easyMQTT_session_chart != session){
          clearInterval(timerUpdateEasyMQTT);
          resetEasyMQTTCharts();
        }
        easyMQTT_session_chart = session;
        $("#easymqtt-config").show();
        setEasyMQTTUpdate($('#easymqtt-update-freq').val());
      }
    }

    function resetEasyMQTTCharts(){
      for (topic in MQTTCharts){
```

```
        MQTTCharts[topic].destroy();
        delete MQTTCharts[topic];
        $('#easymqtt-'+topic).remove();
      }
    }

    function setEasyMQTTUpdate(interval){
      if (typeof interval === 'undefined' || interval < 0){
        alert("Invalid update interval");
        return;
      }
      clearInterval(timerUpdateEasyMQTT);
      updateEasyMQTTJson();
      timerUpdateEasyMQTT = setInterval(updateEasyMQTTJson,interval*1000);
    }

    function dismissEasyMQTTError(){
      $('#easymqtt-warning').html("");
      $('#easymqtt-warning').hide();
    }

    function errorEasyMQTT(error){
      $('#easymqtt-warning').html(error);
      $('#easymqtt-warning').show();

    }

    function clearEasyMQTTSession(){
      if (confirm("Are you sure you want to clear all data in this topic?\nThis
action can not be reverted") == true){
        $.get( "http://bipes.net.br/easymqtt/clearsession.php",
          {"session": easyMQTT_session_chart},
          function( data ) {
            if (data.success){
              resetEasyMQTTCharts();
              errorEasyMQTT(data.result);
            }else{
              errorEasyMQTT(data.result);
            }
          }, "json")
        .fail(function(xhr, status, error){
            errorEasyMQTT("Error clearing session. " + status + " " + error);
          }
        );
      }
    }

    function publishEasyMQTT(topic,value){
      if (typeof topic === 'undefined' || typeof value === 'undefined' ||
topic.length < 1){
        alert("Invalid input");
        return;
      }
      $.get( "http://bipes.net.br/easymqtt/publish.php",
        {"session": easyMQTT_session_chart,
         "topic": topic,
         "value": value
        },
```

```
      function( data ) {
        errorEasyMQTT(data.result);
      }, "json")
    .fail(function(xhr, status, error){
        errorEasyMQTT("Error publising data. " + status + " " + error);
      }
    );
  }

  function addNewChart(topic){
    MQTTCharts[topic] = new Highcharts.stockChart({
      chart: {
        renderTo: 'easymqtt-'+topic,
      },
      time: {
        useUTC: false
      },
      title: {
        text: 'Live data from ' + topic
      },
      rangeSelector: {
        buttons: [{
            count: 1,
            type: 'minute',
            text: '1M'
        }, {
            count: 5,
            type: 'minute',
            text: '5M'
        }, {
            count: 1,
            type: 'hour',
            text: '1H'
        }, {
            count: 12,
            type: 'hour',
            text: '12H'
        }, {
            type: 'all',
            text: 'All'
        }],
        inputEnabled: false,
        selected: 4
      },
      exporting: {
          enabled: true
      },
      xAxis: {
        type: 'datetime',
      },
      yAxis: {
        title: {
          text: 'Value'
        }
      },
      series: [{
        name: topic,
        data: [],
```

```
          marker: {
            enabled: true,
            radius: 2
        },
        }]
      });
    }

    function updateEasyMQTTTopic(topic,since=0){
      $.get( "http://bipes.net.br/easymqtt/gettopic.php",
        {"session": easyMQTT_session_chart, "topic": topic, "since": since},
        function( data ) {
          if (data.success){
            if (data.result.length > 0){
              if (!(topic in MQTTCharts)){
                $("#easymqtt-charts").append('<div id="easymqtt-'+topic+'"
class="easymqtt-chart card"></div>')
                addNewChart(topic);
                for (i in MQTTCharts){
                  MQTTCharts[i].reflow();
                }
              }
              for (i in data.result){

MQTTCharts[topic].series[0].addPoint([data.result[i].timestamp*1000,parseFloat(da
ta.result[i].data)],false);
              }
              MQTTCharts[topic].redraw();
            }
          }else{
            errorEasyMQTT(data.result);
          }
        }, "json")
      .fail(function(xhr, status, error){
          errorEasyMQTT(topic + " request error. " + status + " " + error);
        }
      );
    }
    function updateEasyMQTTJson(){
      //$(".highcharts-navigator").removeAttr("visibility"); //needed only if not
used as an iframe in the main page
      $.get( "http://bipes.net.br/easymqtt/getsession.php",
      {"session": easyMQTT_session_chart},
      function( data ) {
        if (data.success){
          dismissEasyMQTTError();
          for (i in data.result){
            if (data.result[i] in MQTTCharts){
              let last_update =
MQTTCharts[data.result[i]].series[0].points[MQTTCharts[data.result[i]].series[0].
points.length-1].x / 1000;
              updateEasyMQTTTopic(data.result[i],last_update + 1);
            }else{
              updateEasyMQTTTopic(data.result[i]);
            }
          }
        }else{
          errorEasyMQTT(data.result);
```

```
        }
      }, "json" )
      .fail(function(xhr, status, error){
          errorEasyMQTT("Session" + session + " request error. " + status + " " +
error);
        }
      );
    }
  </script>
</body>
</html>
```

APÊNDICE D – Código gerado para validação do módulo MQTT

```
import network
import time
import machine
import dht
import umqtt.robust

data_bytes = None
topic = None

def mqtt_callback(topic,data_bytes):

 topic = topic.decode()
 if (data_bytes.decode()) == '1':
   machine.Pin((5)).off()
 else:
   machine.Pin((5)).on()


sta_if = network.WLAN(network.STA_IF); sta_if.active(True)
sta_if.scan()
sta_if.connect('FITOTEC','password')
print("Waiting for Wifi connection")
while not sta_if.isconnected(): time.sleep(1)
print("Connected")
dhts=dht.DHT11(machine.Pin((4)));dhts.measure();time.sleep(2)
mqtt_buffer = ""; mqtt_client = umqtt.robust.MQTTClient("umqtt_client", server =
'mqtt.thingspeak.com', port = 1883, user = 'api_user', password = 'api_key');
mqtt_client.connect()
mqtt_client.set_callback(mqtt_callback)
mqtt_client.subscribe('channels/1166446/subscribe/fields/field1/api_read_key')
while True:
 dhts.measure()
 mqtt_buffer += ('field1' + "=" + str((dhts.temperature()))) if not
len(mqtt_buffer) else ("&" + 'field1' + "=" + str((dhts.temperature())))
 mqtt_buffer += ('field2' + "=" + str((dhts.humidity()))) if not len(mqtt_buffer)
else ("&" + 'field2' + "=" + str((dhts.humidity())))
 mqtt_client.publish('channels/1158138/publish/api_write_key',
mqtt_buffer,qos=0); mqtt_buffer = ""
 mqtt_client.check_msg()
 time.sleep(60)
```

APÊNDICE E – Código gerado para validação do módulo EasyMQTT

```
import network
import time
import machine
import dht
import umqtt.robust


sta_if = network.WLAN(network.STA_IF); sta_if.active(True)
sta_if.scan()
sta_if.connect('FITOTEC','pequisisal')
print("Waiting for Wifi connection")
while not sta_if.isconnected(): time.sleep(1)
print("Connected")
dhts=dht.DHT11(machine.Pin((4)));dhts.measure();time.sleep(1)
easymqtt_session = "chocadeira"; easymqtt_client =
umqtt.robust.MQTTClient("umqtt_client", server = "bipes.net.br", port = 1883,
user = "bipes", password = "m8YLUr5uW3T"); easymqtt_client.connect()
print("EasyMQTT connected")
while True:
 dhts.measure()
 easymqtt_client.publish(easymqtt_session + "/" + 'temperatura',
str((dhts.temperature())))
 print("EasyMQTT Publish -
Session:",easymqtt_session,"Topic:",'temperatura',"Value:",str((dhts.temperature(
)))
 easymqtt_client.publish(easymqtt_session + "/" + 'umidade',
str((dhts.humidity())))
 print("EasyMQTT Publish -
Session:",easymqtt_session,"Topic:",'umidade',"Value:",str((dhts.humidity())))
 time.sleep(60)
```

APÊNDICE F – Tabela de sessões no banco de dados do EasyMQTT

Tabela 11 – Sessões no banco de dados do EasyMQTT no dia 27 nov. 2020

Sessão	Quantidade de Tópicos	Quantidade de Mensagens
tzqrfq	4	203.305
chocadeira	2	35.968
5sq0sa	1	13.488
m9feub	1	4.475
r23ckr	6	2.274
oaxm7j	2	1.622
e8h8xs	1	1.244
o5jbgl	1	1.240
94s9g	2	276
1f55cu	2	261
4jspy	1	184
9f2h4	2	159
oknas	3	148
66jdr3	1	138
pbes2w	2	133
1c64k	1	61
1kcyve	3	55
769lzw	3	40
2txyl8	1	33
12345	1	31
m5qkn	2	28
nio7u8	3	25
i2cyze	1	24
jg3ge6	2	24
dg05xj	1	20
n7zn5	1	20
dqefu8	1	15
y816q	1	13
fb6vkw	1	11

amoej	1	10
e30lhm	3	10
xpy7ze	1	9
y87qz	1	9
ikpioh	1	8
caio123	2	6
ys7ohi	1	6
admk8	1	5
h40a95	1	5
yc3h5h	1	5
2oe8p8	2	4
6jl5jp	1	4
config	1	4
k4i64	1	4
tzqrfqk	2	4
nio7u7	1	3
p2j3sk	1	3
03vhye	1	2
0dttd8	1	2
adsd21	1	2
g9psa	1	2
n1uybl	1	2
123	1	1
2zcts	1	1
7ao5a	1	1
admin	1	1
iungcn	1	1
local	1	1
rappcn	1	1
Total	87	265.431

Fonte: Elaborado pelo autor.