RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 4 Nro. 1 2023

IoT CLOUD PARA A INTERFACE DO MAGIC MIRROR:
uma abstração da plataforma Blynk para a interface do Magic Mirror

Gabriel Rios Tornich
Graduado em Ciências da Computação – Uni-FACEF
ttornich@outlook.com

Lucas Henrique Couto Cintra
Graduado em Sistemas da Informação – Uni-FACEF
lucascintrac7@gmail.com

Carlos Eduardo de França Roland
Docente do Uni-FACEF
roland@facef.br

Resumo

A comunidade da Internet das Coisas (ou *Internet of Things* - IoT) é crescente e oferece diferentes formas de construir projetos e simplificar etapas que poderiam ser complexas. É nesse sentido que aparecem, por exemplo, as plataformas de IoT *Cloud* e *frameworks* IoT, como o MagicMirror². As plataformas de IoT *Cloud* visam criar uma camada de abstração para comunicação com microcontroladores, como o conhecido Arduino, enquanto o *framework* MagicMirror² oferece uma estrutura desenvolvida na linguagem de programação Electron, com sintaxe facilitada e uma lógica modular voltada para a criação de soluções para espelhos inteligentes. O trabalho descrito neste artigo tem como objetivo desenvolver um módulo MagicMirror² que abstraia funções de uma plataforma de IoT *Cloud* para a interface do espelho inteligente, possibilitando e facilitando interações entre ele e microcontroladores como, por exemplo, o Arduino e Esp8266. Assim sendo possível criar botões, *sliders* e gráficos de monitoramento para comunicação e interação com microcontroladores. Portanto, inicialmente foram analisados os módulos existentes para o *framework*, inclusive alguns que propõe implementar algum tipo de integração com o ambiente Arduino. Foi analisado que nenhum deles permite interagir e controlar microcontroladores diretamente na interface do espelho inteligente. Sendo assim, iniciou-se o trabalho de pesquisa para o desenvolvimento. Inicialmente pesquisou-se plataformas que facilitassem a comunicação com microcontroladores. Foi então selecionada a Blynk para intermediar e facilitar a comunicação entre o espelho e os microcontroladores. Por fim, para testes, construiu-se um protótipo físico de um espelho inteligente, utilizando como microcontroladores um módulo NodeMCU Esp8266 e outro módulo NodeMCU Esp32, que contam com *Wifi* integrado. A partir da realização do desenvolvimento, o trabalho resultou então em um novo módulo para MagicMirror² capaz de abstrair na tela do espelho sensores e controladores interligados com uma placa microcontroladora, via plataforma IoT em nuvem. Os resultados alcançados possibilitam aos usuários dos espelhos inteligentes, um ambiente de controle de itens de casa como luz, temperatura, entre outros dispositivos conectados aos microcontroladores.

Palavras-chave: Internet das Coisas; IoT *Cloud*; MagicMirror; Espelho inteligente

Abstract

The Internet of Things (IoT) community is growing and offers different ways of building projects and simplifying steps that could otherwise be complex. It is in this sense that, for example, IoT Cloud platforms and IoT frameworks, such as MagicMirror², appear. IoT Cloud platforms aim to create an abstraction layer for communication with microcontrollers, such as the well-known Arduino, while the MagicMirror² framework offers a structure developed in the Electron programming language, with easy syntax and modular logic aimed to create intelligent mirror solutions. The work described in this article aims to develop a MagicMirror² module that abstracts functions from an IoT Cloud platform for the smart mirror interface, enabling and facilitating interactions with microcontrollers such as, for example, Arduino and Esp8266. It is possible to create buttons, sliders and monitoring graphics for communication and interaction with microcontrollers. Therefore, initially the existing modules for the framework were analyzed, in order to find some that could implement some type of integration with the Arduino environment. It was analyzed that none of them allow interacting and controlling microcontrollers directly in the smart mirror interface. Therefore, more research for development was done. Initially, platforms that facilitated communication with microcontrollers were studied and Blynk was then selected to intermediate and facilitate communication between the mirror and the microcontrollers. Finally, for testing, a physical prototype of an intelligent mirror was built, using as microcontrollers a NodeMCU Esp8266 module and another NodeMCU Esp32 module, which have integrated Wi-Fi. After carrying out the development, the work then resulted in a new module for MagicMirror2 capable of abstracting sensors and controllers interconnected with a microcontroller board onto the mirror screen, via the IoT cloud platform. The results achieved provide users of smart mirrors with an environment to control household items such as light, temperature, among other devices connected to microcontrollers.

Keywords*: Internet of Things; Smart Mirror; MagicMirror; IoT Cloud;*

1 Introdução

É fato que desde 1999, quando Kevin Ashton cunhou o termo Internet das Coisas (IoT da definição em inglês *Internet of Things*), os projetos e dispositivos disponíveis estão se tornando mais populares e ganhando cada vez mais espaço no mercado, além de muitas comunidades se formando (SAMORA, 2020). Foi nesse contexto que surgiu o projeto *open source* MagicMirror². Como o nome sugere, o projeto encontra-se em sua segunda versão que é disponibilizada como um *framework* para desenvolvimento de interfaces de espelhos inteligentes. *Frameworks* são, em tradução livre, “uma ferramenta que provê componentes prontos ou soluções que são customizáveis para acelerar o desenvolvimento de produtos digitais” (SHARMA, 2023). No caso do MagicMirror², sua base de desenvolvimento é caracterizada por ser modular, ou seja, é possível adicionar novas funcionalidades ao espelho inteligente por meio de módulos.

Por ser um projeto de código aberto, sua comunidade se tornou muito sólida e vem crescendo cada vez mais, juntamente com o desenvolvimento de novos módulos.

É nesse sentido que o projeto apresentado neste artigo se desenvolve. A partir do interesse no *framework* e a possibilidade de colaboração com a comunidade, foram analisados os módulos que já haviam sido criados para ser identificada uma oportunidade de desenvolver algo relativamente novo. Com a ideia de gerenciar microcontroladores a partir de interações na interface do MagicMirror[2], pôde-se concluir que há projetos semelhantes já criados, porém não há nenhum com a proposta das funcionalidades definidas e implementadas neste projeto.

Assim, este artigo apresenta o desenvolvimento de um módulo para ser adicionado ao *framework* MagicMirror[2]. Como requisito, o módulo busca abstrair as funcionalidades da plataforma Blynk de IoT *cloud* para a interface do espelho inteligente, tornando possível que uma aplicação se comunique com microcontroladores através dela. Todos os conceitos necessários para a compreensão do projeto são explicados ao longo do texto, desde a apresentação do ambiente que envolve o termo IoT, o funcionamento do *framework* e da plataforma Blynk, até a conclusão da primeira versão do módulo.

2 *Internet of Things* (IoT)

Esta seção apresenta os fundamentos teóricos para entender a estruturação do projeto, sendo eles o conceito e a história da IoT, área da computação na qual o projeto se desenvolve e o conceito e as características dos microcontroladores. Posteriormente é explicado como estes tópicos se integram ao projeto.

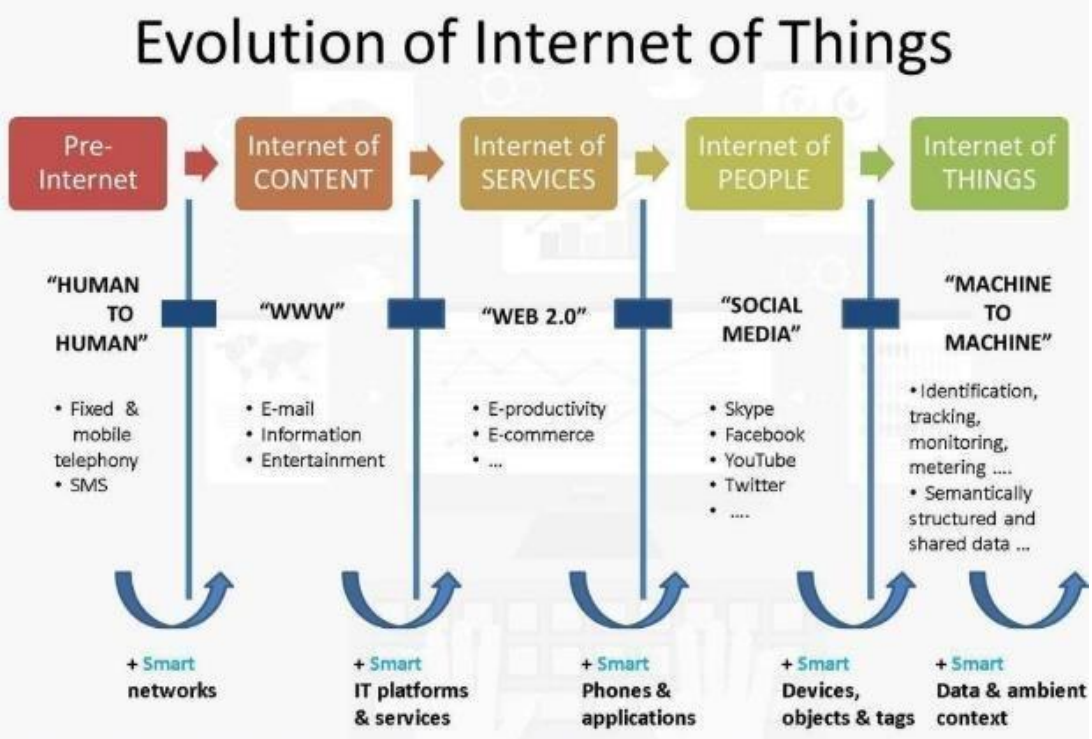
2.1 Definição e história

IoT é um termo abrangente e por vezes pouco entendido em seu real escopo, principalmente pelo público menos inserido no setor tecnológico. Uma definição padrão adotada recentemente é, em tradução livre, “uma rede abrangente de objetos inteligentes que têm a capacidade de se auto-organizar, compartilhar dados e recursos, agindo e reagindo frente a situações e mudanças no ambiente” (MAKADAN, RAMASWAMY, e TRIPATHI, 2015). Ou seja, de forma geral, o conceito caracteriza-se pela conexão de objetos do cotidiano com a internet. Esses objetos unidos a sensores, sistemas embarcados e conexões com a rede, são capazes de coletar dados que podem ser armazenados e processados para gerar informações e saídas programadas. Uma breve representação da evolução da IoT é representada na Figura 1.

Quanto à sua origem, a provável primeira aplicação se dá no ano de 1982, quando programadores que trabalhavam muitos andares acima do de uma máquina de refrigerantes conseguiram escrever um programa que monitorava há quanto tempo um espaço da máquina tinha sido esvaziado. Eles conectaram a máquina na internet para depois determinar, baseado no tempo em que um espaço da máquina estava preenchido ou não, se haveria refrigerante e se estaria suficientemente gelado para compensar a trajeto de vários andares abaixo. Há também a torradeira automatizada que ficou bastante popular e que podia ser ligada e desligada pela internet. Ela foi apresentada durante a INTEROP Conference, por Simon Hackett e John Romkey no ano de 1989 (NEGREIROS, 2020). Contudo o termo *Internet of Things* (Internet das Coisas) foi usado pela primeira vez em 1999 pelo co-fundador do Auto-ID Center no Massachusetts Institute of Technology (MIT) durante uma apresentação para a Procter & Gamble (P&G). Nesta ocasião, ele tratou de um sistema global padronizado para identificação por radiofrequência (RFID) e outros sensores. Assim, utilizou o

termo para definir sistemas em que o mundo físico está conectado à internet via sensores presentes em dispositivos (BHAT, BHAT, GOKLAHE, 2018).

Figura 1 – Evolução da Internet das Coisas



Fonte: CLICTEST (2018)

A área continuou com novos marcos importantes, como a geladeira conectada à internet trazida ao mercado pela LG, mas ainda não houve grande aderência pelos consumidores. O próximo grande marco que veio a popularizar e difundir a IoT, foi a disponibilidade de sistemas microcontrolados mais acessíveis.

2.2 Microcontroladores

Um microcontrolador pode ser definido como um pequeno computador em um único circuito integrado. Um único *chip* que integra todos os recursos de hardware de arquitetura e organização de computadores, designado *System on a Chip* (SOC). Geralmente suas aplicações estão dentro de um sistema embarcado para controle e automatização de processos (LUTKEVICH, [s.d.]). O primeiro microcontrolador comercialmente disponível foi o TMS 1000. Ele foi apresentado em 1971 e passou a ser vendido em 1974, combinando processador, *clock* interno, memória de leitura/gravação e memória de somente leitura em um único *chip* e tinha como alvo sistemas embarcados.

> As funcionalidades agregadas aos microcontroladores inicialmente se resumiam a interfaces de entrada e saída (I/O) e foram agregando, a cada nova versão ou produto, memória RAM, memória EPROM para programas e dados e circuito de oscilador (clock), interfaces de comunicação (serial, USB), e mais recentemente, interfaces de rede Ethernet, WiFi e Bluetooth. Um

> microcontrolador com todas essas funcionalidades integradas é capaz de atender a um número de aplicações sem precedentes (OLIVEIRA, 2017).

É nesse sentido da evolução da tecnologia e por consequência das plataformas microcontroladas, como o ESP8266 e o Arduino, que foi gerado grande impacto no segmento de IoT. O Arduino surgiu em 2005, no Interaction Design Institute Ivrea (IDII), na cidade de Ivrea, na Itália, a partir da necessidade de um professor da disciplina de Design e Interação diminuir os custos para desenvolvimento de projetos acadêmicos com eletrônica embarcada feitos à época. Sendo uma iniciativa de sucesso, foi reconhecida, obteve menção honrosa na categoria de Comunidades Digitais no ano de 2006, e atingiu a marca de cinquenta mil placas vendidas em outubro de 2008 (COSTA, [s.d.]). O Arduino trata-se, na verdade, não de um microcontrolador especificamente, mas sim de uma “plataforma de desenvolvimento de sistemas embarcados de baixo custo aberta e livre. Assim não está vinculado a nenhum fabricante específico, embora a maioria dos módulos disponíveis utilize microcontroladores da Atmel*”* (OLIVEIRA, 2018). O Arduino oferece um ambiente de desenvolvimento com uma *Integrated Development Enviroment* (IDE) e uma boa gama de placas de circuito, como o Arduino Uno, por exemplo, que tem valor de venda na faixa de 60 a 80 reais, se referindo ao mercado brasileiro.

Pode-se afirmar que esse foi um grande marco, pois a partir daí o desenvolvimento no universo de sistemas embarcados e de IoT tornou-se acessível a desenvolvedores, estudantes e entusiastas da área, que com um módulo relativamente barato e uma plataforma de desenvolvimento, puderam desenvolver soluções e alcançar grandes avanços no segmento de automação de processos em Internet das Coisas.

3 Espelho Inteligente

O cenário de casas inteligentes vem se tornando cada vez mais acessível, principalmente com a popularização dos assistentes pessoais, como o Google Home da Google, e o Echo Dot da Amazon. De forma geral, os usuários desses dispositivos enviam comandos por meio da voz e os aparelhos, que contam com uma inteligência artificial capaz de controlar dispositivos compatíveis, interpretam e realizam as devidas ações. Assim, cria-se um ambiente inteligente no qual não só *smartphones* ou computadores, mas também cortinas, luzes e portas, podem estar conectados à rede e responder aos comandos do morador. É nesse sentido de artigos domésticos inteligentes que aparece o conceito do espelho inteligente ou *smart mirror,* em inglês

À época desse estudo, já se encontram casas com espelhos que possuem botões que permitem ligar ou desligar uma luz de fundo, por exemplo. Porém o conceito tem evoluído juntamente com os avanços e popularização do desenvolvimento de IoT e já se tem trabalhado para que esse artefato possa agregar informações úteis na tela, além de funcionalidades por meio de interação. S*mart mirrors* mais modernos e com mais recursos já existem no mercado, porém ainda são muito caros, podendo ser encontrados na faixa de 2 a 6 mil reais. Por essa causa, estão ainda distantes de se tornar um recurso popular.

Atualmente a maior difusão e exploração deste conceito parte não das grandes empresas, mas sim da crescente comunidade de desenvolvimento IoT. Os *smart mirrors* parecem chamar muito a atenção dos usuários. Plataformas, *frameworks* e padrões de construção já têm se moldado e ganhado popularidade, formando uma comunidade latente, como é o caso do *framework* MagicMirror², objeto de estudo deste artigo.

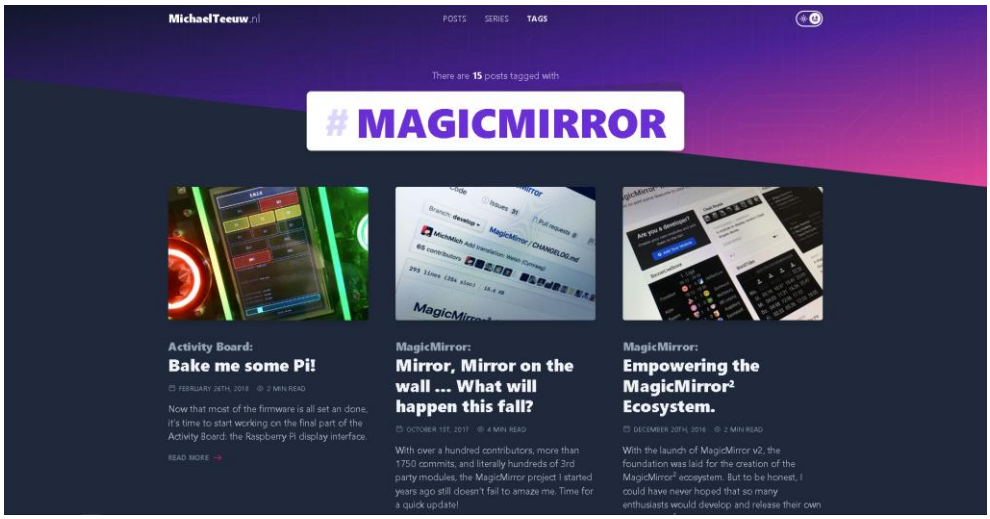
A construção de espelhos inteligentes depende, de forma geral, do uso do espelho de duas vias. Trata-se de uma superfície capaz tanto de refletir como deixar passar um pouco de luz. Assim, caracteriza-se por ser um espelho comum, porém com um pouco de transparência, que possibilita que o usuário enxergue luzes emitidas atrás, onde estará a tela com o sistema responsável pelas funcionalidades a serem agregadas. As possibilidades são infinitas e podem ser exploradas de acordo com o contexto de instalação do espelho, seja numa casa, num ambiente comercial ou acadêmico.

4 O *framework* MagicMirror²

A ideia do MagicMirror foi desenvolvida por Michael Teeuw que inicialmente não tinha a intenção de ser um *framework* como base para desenvolvimentos. Segundo o relato que ele apresenta em seu blog, a ideia surgiu quando passeava com sua namorada pela loja Macy‘s, em Nova York, quando viu um espelho com um canto iluminado e decidiu que iria construir seu próprio espelho mágico. Em seu blog (Figura 2), o criador conta detalhadamente como foi o desenvolvimento da ideia até o modelo que está disponível hoje de forma *open source*.

Durante a construção do espelho inteligente, o criador publicava em seu blog os resultados do seu desenvolvimento, o que começou a gerar interesse na comunidade para construírem seus próprios *magic mirrors*. A comunidade foi crescendo rapidamente e, com isso, funcionalidades foram sendo criadas e adicionadas ao espelho, mas para isso necessitava editar o código fonte do sistema, além de que, para executar a aplicação, era preciso configurar um servidor Apache. Nesse momento a aplicação era desenvolvida para ser executada apenas fazendo o uso do microcomputador RaspberryPi.

Figura 2 – Blog de Michael Teeuw, criador do MagicMirror



Fonte: os autores

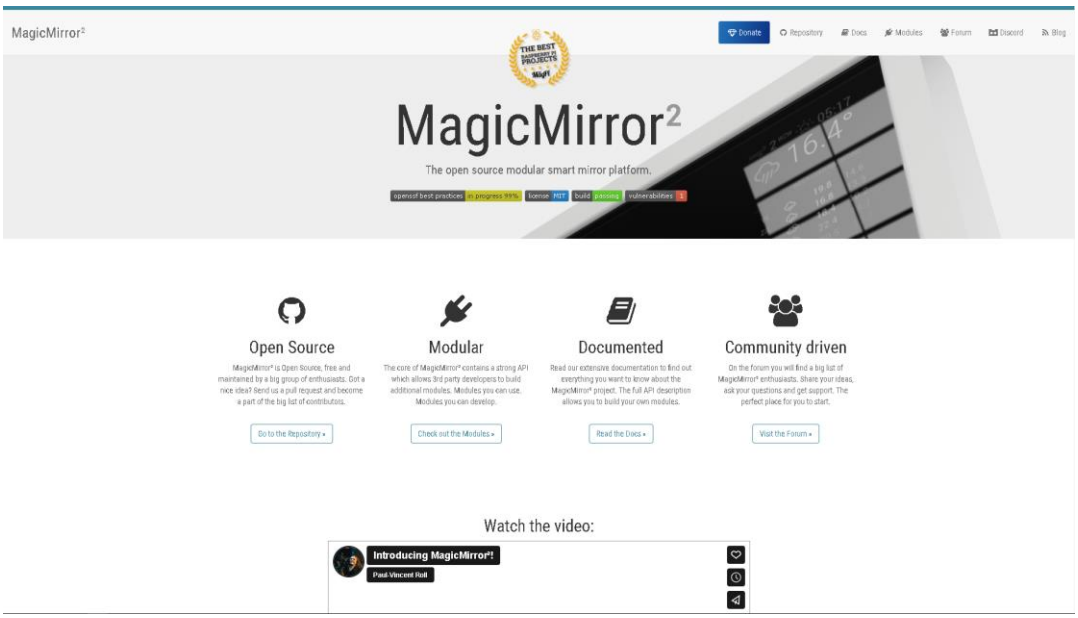
O projeto tomou melhor forma com o lançamento do MagicMirror^2. Devido à popularização surpreendente, até para o criador, que identificou a necessidade de ajustar a plataforma de uma maneira que seja fácil de ser instalada, ser executada praticamente em qualquer lugar e ser escalável, criando o sistema de módulos. Assim, passou a não ser mais necessário editar o código fonte para adicionar novas

funcionalidades no MagicMirror^2, mas sim apenas adicionar novos módulos. Essas modificações para lançar a segunda versão do projeto foram feitas com a ajuda de membros da comunidade. Além disso, como parte fundamental de um projeto, criaram a documentação detalhada, mostrando os processos de instalação, de criação e desenvolvimento de novos módulos e organizaram os módulos desenvolvidos pela comunidade em um repositório no GitHub.

Atualmente a versão MagicMirror^2 é a que está disponível de forma *open source* em seu site: https://magicmirror.builders, podendo ser facilmente explorada e utilizada de forma gratuita (Figura 3).

Como citado anteriormente, os módulos da comunidade foram organizados no GitHub e classificados de acordo com sua funcionalidade, como pode ser visto na Figura 4.

Figura 3 – MagicMirror^2



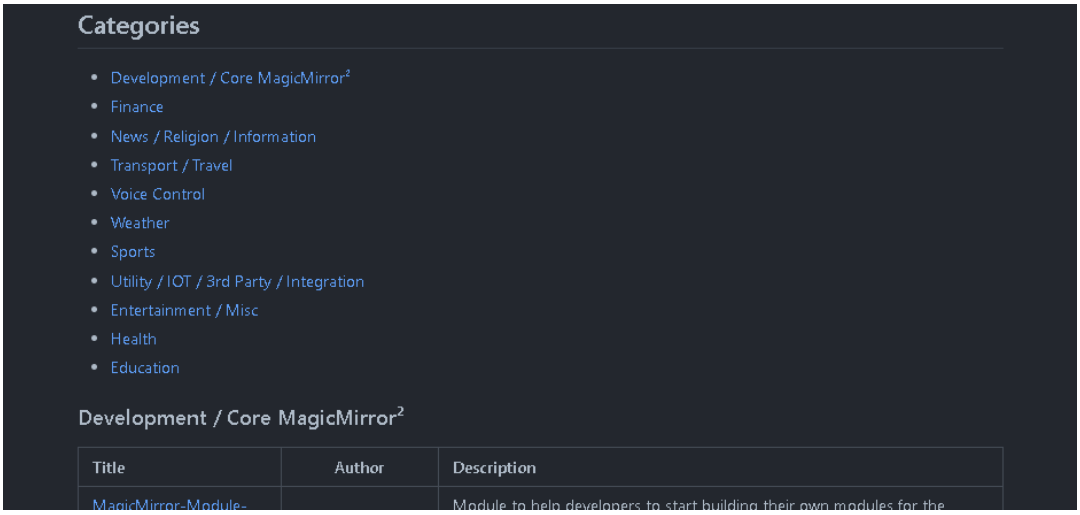
Fonte: os autores

5 Módulo proposto

Com base na análise dos módulos desenvolvidos pela comunidade, disponíveis no repositório do GitHub, buscou-se por módulos que oferecessem comunicação e interação fácil entre o espelho e microcontroladores, como o Arduino e o Esp8266.

Foram encontrados dois módulos que se assemelham ao objetivo da busca. Um deles, chamado MMM-ArduPort, oferece suporte de comunicação serial do Arduino com RaspberryPI, na qual o Arduino envia dados de sensores para o módulo usando Python. O outro, chamado MMM-ObjectBlocks, oferece suporte para a comunicação entre o Magic Mirror e o Arduino via Web Socket e ObjectBlocks *IoT Education Platform*.

Figura 4 – GitHub e módulos do MagicMirror^2



Fonte: os autores

Pontos negativos que pesaram na análise destes módulos foram que a comunicação do primeiro módulo apresentado depende de uma ligação física entre o Arduino e RaspberryPi. Outra limitação deste é justamente que essa comunicação funciona somente entre estes dispositivos, não permitindo um cenário no qual o Arduino, ou outro microcontrolador, esteja distante do espelho. Já no segundo módulo, não foi encontrada uma documentação para a utilização correta da plataforma ObjectBlocks.

Levando em consideração os pontos apresentados, como proposta de solução, foi iniciado o desenvolvimento de um novo módulo utilizando o *framework* MagicMirror^2. O módulo tem como objetivo possibilitar e facilitar interações entre o espelho inteligente e microcontroladores como, por exemplo, o Arduino e Esp8266. Para estabelecer a comunicação do espelho com os microcontroladores foi feito o uso da plataforma de IoT *cloud* Blynk. A Blynk oferece um ambiente *cloud* robusto, com documentação ampla e simplificada, além de uma API para desenvolvimento externo com a plataforma.

API se trata, em tradução livre, de “um acrônimo para *application programming interface* (interface de programação de aplicação), um software intermediário que permite duas aplicações comunicarem entre si” (FRYE, [s.d.]).

Sendo assim, a Blynk foi responsável por intermediar a comunicação entre o espelho inteligente e os microcontroladores. Ao longo dessa seção serão discutidos os pontos positivos que levaram à escolha do uso da Blynk (pois existem outras plataformas com funcionalidades semelhantes), além de explicar como foi utilizada no projeto e o motivo de utilizar uma plataforma para intermediar a comunicação.

5.1 Plataformas de IoT

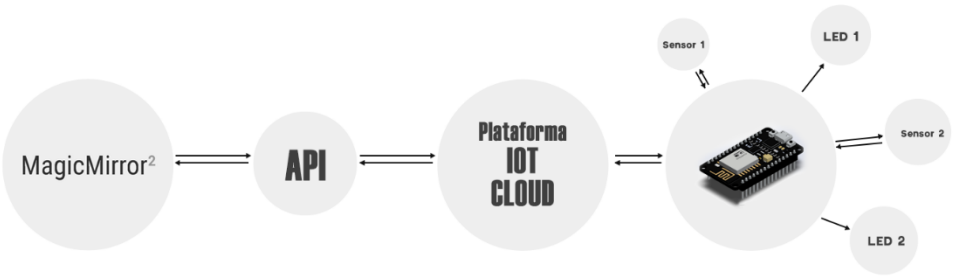
Primeiramente é importante explicar o motivo de ser utilizada uma plataforma de IoT *cloud* para intermediar a comunicação entre o espelho inteligente e os microcontroladores. Essas plataformas oferecem estrutura pronta de comunicação com microcontroladores compatíveis, sendo possível a utilização de botões virtuais, gráficos para monitoramento, entre outros recursos. Para utilização, cada plataforma oferece uma estrutura de código com suas devidas bibliotecas, bastando cadastrar um microcontrolador na plataforma, carregar o código base no dispositivo, ajustar as credenciais de comunicação geradas e configurar os botões e recursos que serão

utilizados. Ou seja, é menor a necessidade de código e configurações a serem desenvolvidos manualmente.

Tendo a parte da comunicação pronta, é preciso de uma forma para utilizar os recursos da plataforma de IoT *cloud* por meio da interface MagicMirror[2]. Para isso, as plataformas geralmente oferecem APIs que possibilitam consultar e manipular valores de variáveis que estão ligadas aos recursos da plataforma.

Exemplificando, com a API é possível fazer uma requisição para alterar o valor de uma variável que está ligada a um botão que liga ou desliga um led. Assim, os recursos da plataforma podem ser utilizados externamente. Na Figura 5 é mostrada a estrutura de comunicação do projeto.

Figura 5 – Esquema de comunicação do projeto



Fonte: os autores

Em relação às plataformas disponíveis no mercado, as que se encaixam mais com a proposta do projeto são a Arduino IoT Cloud e a Blynk. As duas se assemelham muito às funcionalidades, interface e estruturação. Em ambas é possível configurar dispositivos microcontroladores e enviar e receber dados por meio dos *widgets*, que são os recursos que a plataforma oferece, sendo eles botões, gráficos, botões deslizantes, entre outros.

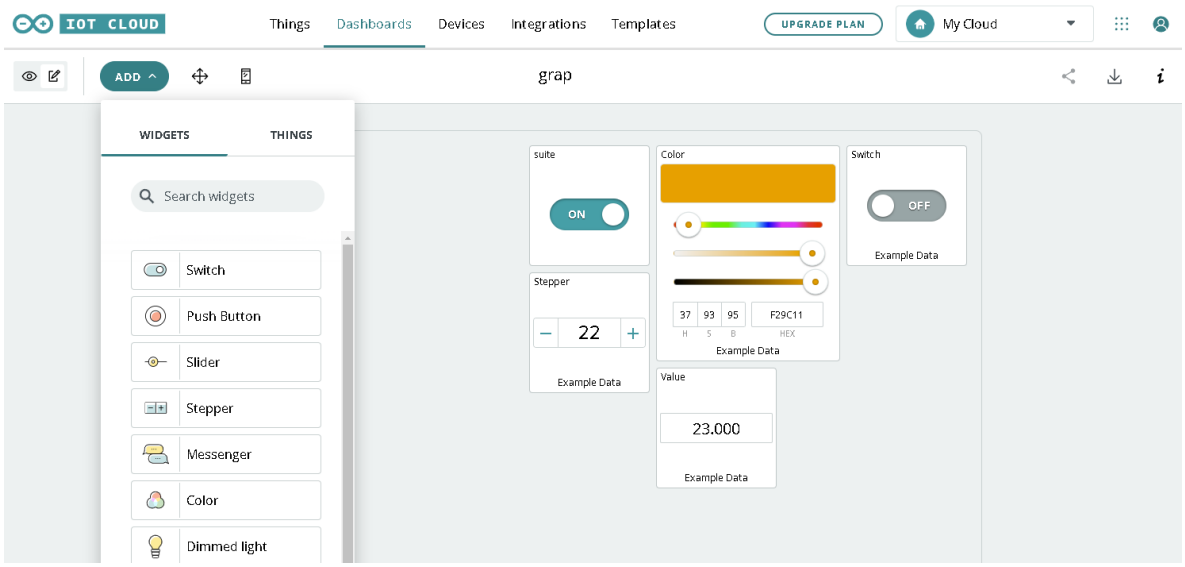
Além disso, ambas disponibilizam serviço de API para integração com desenvolvimento externo de soluções que se comuniquem com a plataforma.

Na Figura 6 está apresentada a interface da plataforma Arduino IoT Cloud e, na Figura 7 a interface da plataforma Blynk.

Para o objetivo do projeto optou-se por utilizar a plataforma Blynk, por oferecer uma API de mais simples integração, documentação mais robusta e gratuita em grande parte, sendo pagos alguns recursos.

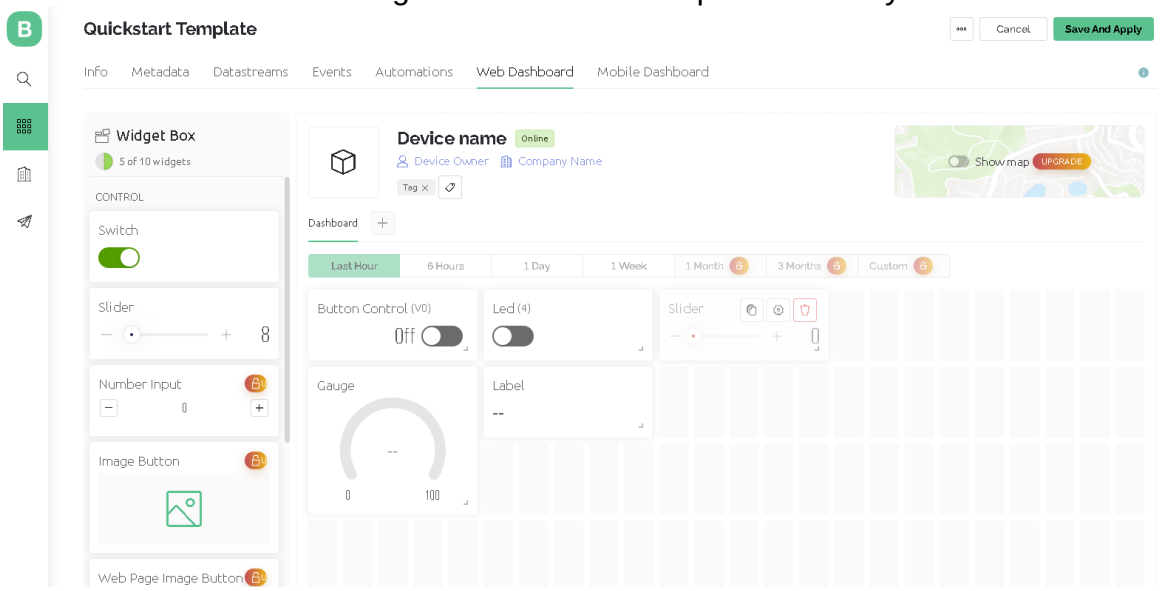
Concluindo a estruturação do projeto, para desenvolvimento do módulo com o *framework* MagicMirror[2] utilizou-se a API da plataforma Blynk para abstrair seus recursos para a interface do espelho inteligente, sendo a Blynk, intermediadora da comunicação com os microcontroladores.

Figura 6 – Interface da plataforma Arduino IoT *Cloud*



Fonte: os autores

Figura 7 – Interface da plataforma Blynk



Fonte: os autores

6 Requisitos

Tendo estabelecido as ferramentas e a estrutura do projeto para desenvolvimento, definiram-se os requisitos da solução. Como requisitos funcionais o módulo deve:

- Permitir criar e personalizar *widgets* da plataforma Blynk na interface do MagicMirror², como também configurar variáveis para funcionamento correto da API;

- Inicialmente devem ser mapeados os seguintes *widgets*: *button* (*botão*), *slider* (controle deslizante*)*, *label* (rótulo, ou legenda), *gauge* (medidor) e *device status* (status do dispositivo).
- Permitir configurar dispositivos microcontroladores conectados à plataforma.

E como requisitos não funcionais o módulo deve:

- Ser fácil de instalar e utilizar;
- Manter sincronização com os recursos da plataforma Blynk;

7 Metodologia

Como metodologia de gestão do projeto, foi utilizado o SCRUM, durante os períodos de seis meses, divididos entre *sprints* de duração entre três a quatro semanas. Durante o primeiro semestre de 2023 foi desenvolvido o escopo geral deste projeto, com acompanhamento do docente do curso e orientador deste trabalho. Finalizado esse período, o decorrer e acompanhamento do desenvolvimento seguiu-se através de reuniões semanais através do Google Meets, para alinhamento e divisão de tarefas entre os autores. Foi também utilizada a plataforma Trello principalmente para acompanhamento do projeto por parte do orientador.

8 Desenvolvimento

Esta seção inicialmente apresenta parte do funcionamento do MagicMirror² e em seguida o desenvolvimento do módulo proposto e sua estrutura de funcionamento.

8.1. Instalação e configuração inicial do Magic Mirror²

Tendo definido a documentação inicial e o ambiente de desenvolvimento, iniciou-se o processo de desenvolvimento.

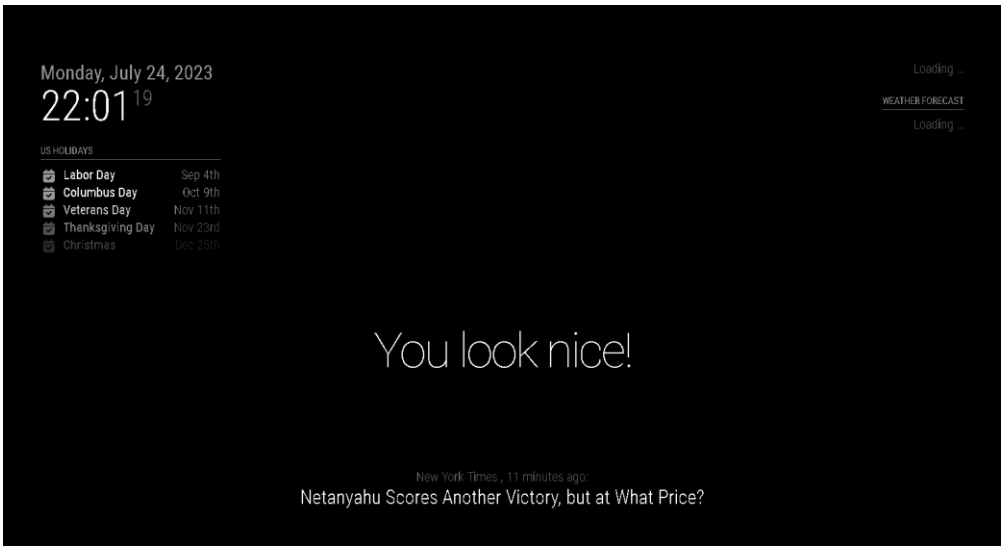
O *framework* utilizado é definido como:

> MagicMirror² é uma plataforma modular e *open source* de espelho inteligente. Com uma lista crescente de módulos instaláveis, o MagicMirror² permite converter sua sala de estar ou banheiro em um assistente pessoal. MagicMirror² é construído pelo criador do MagicMirror original, com a incrível ajuda de uma comunidade crescente de contribuintes.
>
> MagicMirror² foca em um plugin de sistema modular e usa Electron como ambiente de desenvolvimento. Então não é mais necessário servidores web ou instalação de navegadores específicos (MAGIC MIRROR², 2016).

Analisando os requisitos para executar o MagicMirror², verifica-se que ele foi desenvolvido originalmente para RaspberryPi. Por esse motivo, o ambiente ideal para executar a aplicação é em um destes dispositivos instalado na estrutura física do espelho. Porém, é possível executar a aplicação diretamente no Windows, realizando algumas configurações. Essas configurações juntamente com o processo de instalação e o processo e estrutura de como funciona o *framework* estão disponíveis em sua documentação.

Na versão atual do projeto, MagicMirror², alguns módulos vêm instalados por padrão. São eles o *Alert, Calendar, Clock, Compliments, Hello World, News Feed, Update Notification* e *Weather*. Ao executar o MagicMirror² pela primeira vez, tem-se a tela com os módulos padrões como representada na Figura 8.

Figura 8 – Aplicação do MagicMirror² com os módulos padrões



Fonte: os autores

8.2. Estrutura do *framework*

Em relação às configurações dos módulos, existe um arquivo principal que reúne essas definições. É no arquivo ***config.js***, da pasta **config**, que são parametrizados os módulos que serão iniciados com a aplicação, com algumas propriedades a serem configuradas. Por exemplo, a propriedade ***position***, que define em qual posição da tela o módulo será exibido. Podem também ser definidas propriedades específicas para cada necessidade de um módulo. Além disso, a aplicação pode ser iniciada via servidor web, possibilitando ser acessada via navegador ou ainda de forma local, no próprio *desktop*, através de interface Electron. E as configurações de rede para acesso via navegador web, também podem ser alteradas no mesmo arquivo de configuração.

O *framework* também vem com CSS padrão, que pode ser personalizado inserindo um arquivo de CSS no caminho ***css/custom.css***. Como padrão, os módulos são inseridos na pasta ***modules***, com os módulos padrão inseridos na subpasta ***default***. É na pasta ***modules*** que novos módulos podem ser criados ou adicionados. O arquivo principal de cada módulo deve seguir o caminho ***nome-do-modulo/nome-do-modulo.js***. Um módulo pode conter um arquivo auxiliar opcional chamado ***node_helper.js*** que pode ser um intermediário para possíveis comunicações externas

da aplicação. E qualquer outra subpasta pode ser inserida na pasta do módulo para inserção de estilos CSS ou imagens, por exemplo.

8.3. Estrutura do arquivo principal do módulo

Como MagicMirror[2] se trata de um *framework*, o arquivo principal possui uma estrutura padrão para funcionar corretamente definida por um arquivo JavaScript que inicialmente deve ser adicionado o script ***Module.register()***. Como primeiro parâmetro deve ser inserido o nome do módulo, que para o projeto foi definido o nome BlynkCloud, e no segundo é construída toda a parte principal do módulo. Outro recurso padrão que o *framework* oferece é adicionar um objeto chamado ***defaults*** que pode conter algumas configurações iniciais como a definição de variáveis e valores padrão. Porém, na construção do módulo proposto não foi necessária sua utilização.

Para adicionar o módulo na interface do MagicMirror[2], utiliza-se a função **getDom**. Essa função retorna um elemento, chamado ***wrapper***, que contém a implementação das regras no corpo do módulo. Em relação ao módulo que foi desenvolvido, essa função gera um log avisando que o módulo inicializou, cria o elemento *wrapper*, atribuindo-o uma classe para configurar o estilo, e cria outro elemento chamado ***components***. Esse elemento armazena o retorno da função ***Components()***, que como parâmetros, são passadas as configurações necessárias para construir o corpo do módulo. Essas configurações são as que foram definidas no arquivo ***config.js***. Em seguida o corpo do módulo é anexado ao elemento principal, ***wrapper***, para ser carregado na interface da aplicação.

A próxima função do *framework* a ser chamada no projeto é a ***getScripts***. Ela é responsável por incorporar quaisquer *scripts* adicionais que precisem ser usados. Para o projeto, foi criado apenas o arquivo adicional ***widgets.js****,* responsável pela lógica definida no módulo.

Outra função que foi utilizada é a ***getStyles***, que chama para o arquivo principal, um ou mais arquivos de estilos, do tipo ***.css***. Para o projeto foi criado apenas um arquivo de estilo, chamado ***panel.css***.

Assim, conclui-se o desenvolvimento do arquivo principal do módulo. O próximo passo foi desenvolver a lógica principal do projeto, escrita no arquivo ***widgets.js***, citado anteriormente. É nesse arquivo que é estabelecida a comunicação com a API da Blynk e onde foram implementadas as lógicas para mapeamento e interação dos botões e recursos da plataforma. Mas antes é necessário entender como funciona a estrutura da plataforma.

8.4. Funcionamento da plataforma Blynk

Ao ingressar na plataforma, algumas configurações iniciais a partir do modelo *Quickstart* são sugeridas, como a configuração de um dispositivo, seleção da IDE de desenvolvimento e a geração de um modelo de código para carregar em um microcontrolador para estabelecer comunicação com a plataforma. Para compreender a estrutura de comunicação da plataforma com um microcontrolador é preciso entender os conceitos e funcionamento dos *widgets*, *devices*, das variáveis *Datastreams* e como todos estes se interligam.

Partindo do início, é necessário que um dispositivo seja configurado, indicando qual é o tipo de microcontrolador com o qual deseja se comunicar. Um *device* é a representação do microcontrolador físico.

Configurado o dispositivo, tem-se o acesso aos *widgets*. *Widgets* são os recursos que a plataforma oferece para comunicar (enviar e receber dados) com o microcontrolador. Exemplos desses recursos são os botões, *sliders*, gráficos e *labels* para monitoramento, além de outros recursos diferentes. A Figura 7 apresentou a interface de *dashboard* da plataforma com alguns *widgets* selecionados.

Vale ressaltar que não são todos os *widgets* que são disponibilizados na assinatura gratuita. Com os *widgets* já adicionados ao *dashboard*, no modo de edição é possível definir as configurações de cada um.

Um *Datastream* é a variável, criada na plataforma, que vai estabelecer comunicação de um *widget* com um microcontrolador, seja diretamente com um pino físico ou indiretamente através dos *Virtual Pins*. É possível criar *Datastreams* do tipo *Digital Pin, Analog Pin, Virtual Pin, Enumerable* e *Location*, alguns podendo ser de entrada (*input*) ou saída de dados (*output*). No caso dos *Digital* e *Analog Pins* (que têm o recurso de comunicação direta com um pino do microcontrolador), se for criado um *Datastream* atrelado ao pino 4, este irá comunicar com o pino 4 do microcontrolador. Já nos *Virtual Pins*, as mudanças de estado são administradas via variáveis no código do microcontrolador, não diretamente relacionadas aos pinos.

Os *Datastreams*, como explicado, são atreladas aos *widgets*. Então, todas as ações que forem feitas em um *widget*, serão passadas para o microcontrolador através dos *Datastreams*. No cenário atual em que o projeto se desenvolve, os *Datastreams* do tipo *Digital* e *Analog* estão temporariamente suspensos na plataforma. Portanto, para o desenvolvimento e testes do módulo proposto foram utilizados apenas *Virtual Pins*.

A partir desse ponto, tendo entendido a estrutura básica de comunicação da plataforma Blynk, é possível compreender o desenvolvimento do arquivo da lógica do módulo proposto, utilizando a API da plataforma para alterar valores dos *Datastreams* ou ler o estado destas variáveis.

8.5. Desenvolvimento da lógica principal do módulo

De forma geral, o código foi dividido entre as funções que criam os *widgets*, as funções auxiliares e a função *Components()*, já citada, responsável por criar o corpo do módulo de acordo com as configurações definidas.

Como definido no levantamento de requisitos, foram abstraídos os 5 principais *widgets* da plataforma, sendo eles: *device status*, *buttom*, *slider*, *gauge* e *label*. Para sincronizá-los e comunicar com a plataforma Blynk, foram utilizados os *endpoints* da API. *Endpoint* trata-se, em tradução livre, de “local (geralmente URL) de onde APIs podem acessar recursos que elas precisam para realizar alguma função” (SMARTBEAR, 2023).

Em relação à sincronização, todos os *widgets* quando criados, antes de serem utilizados, é necessário realizar uma chamada na API para sincronizá-los com o estado do *Datastream* referente a cada um. Após isso, os *widgets* permanecem monitorando mudanças de estado dos *Datastreams* para sincronização caso sejam alterados pelos *widgets* mapeados na plataforma. Mas também podem fazer chamadas para mudar o valor dos *Datastreams*.

Para entender a relação entre a as funções que mapeiam os *widgets* no MagicMirror[2], as funções auxiliares e os *endpoints* que cada uma utiliza para sincronizar e atualizar valores da plataforma, foi criada a Tabela 1.

Tabela 1 – Abstração das funções do código da lógica do módulo

Função	Função auxiliar	Endpoints da API	
		Para sicronização	Para atualização de valores
createDeviceStatus	**loadStatusDevices**: Responsável por sincronizar o status do dispositivo	https://blynk.cloud/external/api/isHardwareConnected?token=${**token**}	----------
createButton	**loadButton:** Responsável por sincronizar e atualizar o estado do botão com o estado do Datastream referente a ele na plataforma	https://blynk.cloud/external/api/get?token=${**token**}&dataStreamId=${**streamId**}	https://blynk.cloud/external/api/update?token=${**token**}&dataStreamId=${**streamId**}&value=**1**
			https://blynk.cloud/external/api/update?token=${**token**}&dataStreamId=${**streamId**}&value=**0**
createSlider	**loadSlider:** Responsável por sincronizar e atualizar o estado do slider com o estado do Datastream referente a ele na plataforma		https://blynk.cloud/external/api/get?token=${**token**}&dataStreamId=${**streamId**}
			https://blynk.cloud/external/api/update?token=${**token**}&dataStreamId=${**streamId**}&value=${**value**},
createGauge	**loadGauge:** Responsável por sincronizar o status do gauge com o estado do Datastream referente a ele na plataforma		----------
createLabel	**loadLabel:** Responsável por sincronizar o status do label com o estado do Datastrem referente a ele na plataforma		----------

Fonte: os autores

Observa-se que na tabela, não é citado que as funções auxiliares sincronizam e atualizam o estado dos *widgets* do MagicMirror² com o estado dos *widgets* da plataforma Blynk, mas sim com os *DataStreams*. Isso se dá, pois, os *widgets* da plataforma também estão atrelados aos *DataStreams*. Sendo assim, valores e *status,* de ativado ou desativado, por exemplo, são controlados por essas variáveis. Por isso, quando um botão mapeado no MagicMirror² atualiza uma dessas variáveis, se houver um *widget* na Blynk atrelado a ela, o estado dele também é alterado. Dessa maneira, se um *widget* for mapeado no MagicMirror², não é necessário mapear também na plataforma, basta criar e configurar um *Datastream* para que funcione.

Após a criação dos *widgets*, a função *Components*() reúne todos para serem retornados juntamente com o elemento *Wrapper*, para assim serem apresentados na interface do MagicMirror².

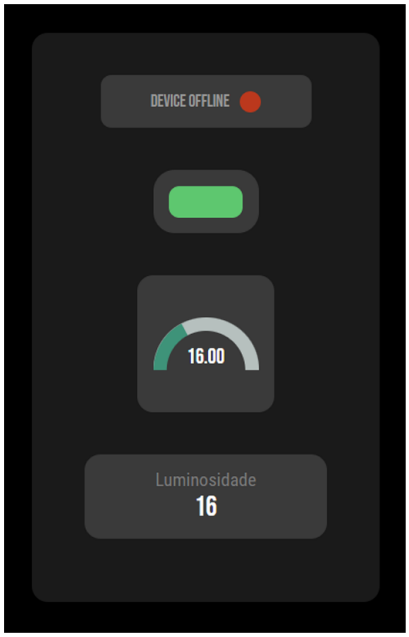
A Figura 9 apresenta um exemplo de configuração para o módulo e seu resultado na interface do MagicMirror² em sincronia com a plataforma.

Figura 9 – Exemplo de configuração do módulo

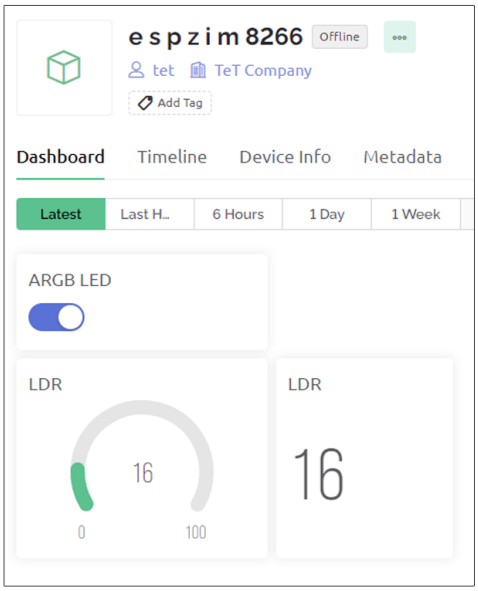
Arquivo de configuração

```
{
      module: 'blynkcloud',
      position: 'top_right',
      config: [
        module_style = [
          width = "270px",
          heigh = "480px",
        ],
        devices = [
          device1 = [
            token = "[redacted]",
            widgets = [
              widget1 = [
                type = "device_status",
                dataStream = "none",
                style = [
                  width = "200px",
                  height = "50px",
                  icon_size = "20px",
                  font_size = "12pt"
                ]
              ],
              widget2 = [
                type = "button",
                dataStream = "1",
                style = [
                  width = "70px",
                  height = "30px",
                ]
              ],
              widget3 = [
                type = "gauge",
                dataStream = "3",
                style = [
                  width = "100px",
                  height = "100px",
                  range_min= "-40",
                  range_max = "150",
                  font_size = "15pt"
                ]
              ],
              widget4 = [
                type = "label",
                dataStream = "3",
                style = [
                  label_name = "Luminosidade",
                  width = "200px",
                  heigh = "50px",
                  name_font_size = "13pt",
                  text_font_size = "20pt",
                  widget_orientation = "center"
                ]
              ],
            ]
          ],
        ]
      ]
    }
```

Interface do Magic Mirror



Interface da plataforma



Fonte: os autores

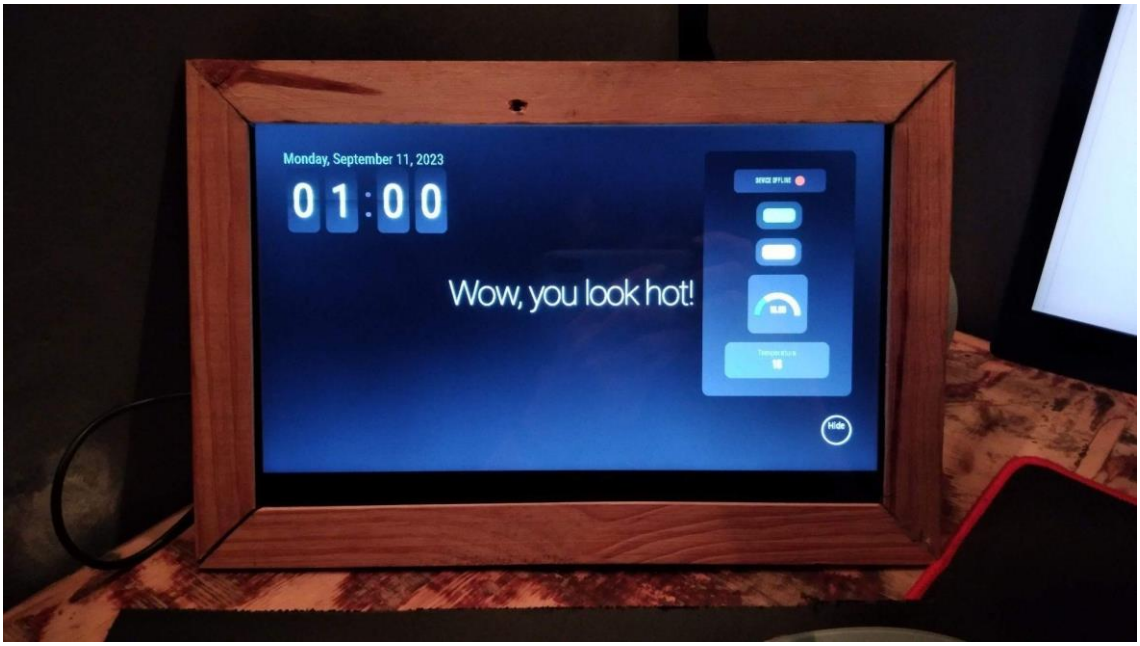
9 Cenário de Testes

Após o desenvolvimento do módulo, foi construído o ambiente de testes. Esta seção descreve os processos realizados e o ambiente criado para os testes, mencionando as configurações e os resultados obtidos.

9.1. Construção do protótipo de um *Smart Mirror*

No planejamento inicial do escopo do projeto descrito neste artigo não foi idealizada a construção do projeto físico do espelho inteligente, pois trata-se de uma construção dispendiosa para o cenário deste trabalho, que tem foco na solução de software. Além disso, o módulo desenvolvido é possível ser testado independente do projeto físico. Porém, no decorrer do desenvolvimento, com a solução se tornando funcional e mais robusta, foi elaborada a construção de um protótipo para tornar os testes mais fidedignos. Foi então construído um protótipo de *Smart Mirror* utilizando um notebook com tela *touch screen*, acoplando suas peças e a tela *touch* em uma moldura. Em seguida foi instalado o MagicMirror[2] juntamente com o módulo desenvolvido. A Figura 10 mostra o resultado da construção.

Figura 10 – Construção física do *Smart Mirror*



Fonte: os autores

9.2. Configuração de microcontroladores

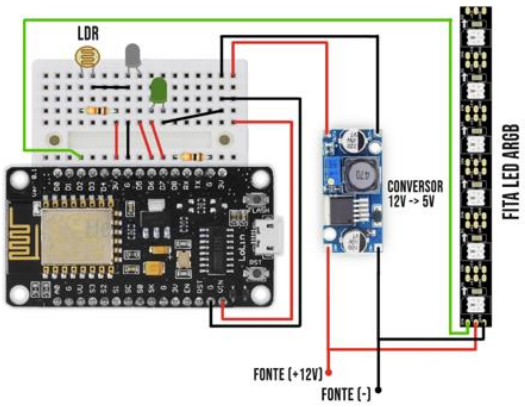
O próximo passo foi configurar os microcontroladores e definir o que eles iriam controlar. Como microcontroladores, foram usados um ESP8266 e um ESP32, que contam com WIFI embutido, facilitando os testes.

> O Módulo NodeMCU V3 - ESP8266 - CH340 ou Nodemcu Lolin é uma placa controladora capaz de colocar seu projeto conectado na internet, assim fazendo com que você seja capaz de acessá-lo, controlá-lo e adquirir informações de forma remota. Em outras palavras, essa é a porta de acesso para a Internet das Coisas (Internet of Things - IoT).
>
> O Módulo NodeMCU V3 - ESP8266 - CH340 é uma placa de desenvolvimento que combina o chip ESP8266 com uma interface usb-serial para comunicação e um regulador de tensão 3.3V em sua placa, podendo ser alimentado com até 9V pela sua entrada micro usb, que também é utilizada para que seja feita sua programação nas linguagens LUA ou a IDE do Arduino (C, C++). Esta pequena e poderosa placa é ideal para estudantes, hobbystas e profissionais da área de eletrônica que possuem interesse em ingressar no mundo da Internet das Coisas (Internet of Things - IoT) (CURTO CIRCUITO, 2016).

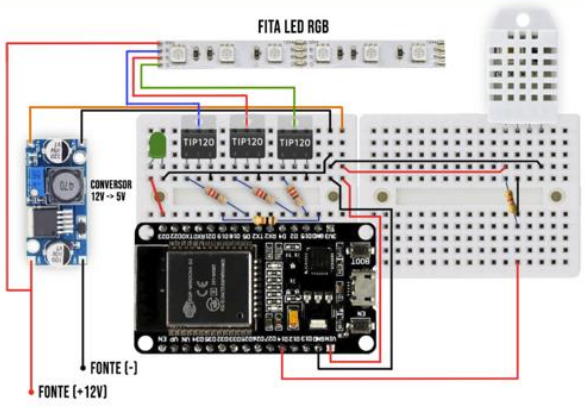
Para esquematizar o cenário com os periféricos e sensores que foram adotados foi utilizado o software Photoshop juntamente com Microsoft Excel para organizar algumas informações. A Figura 11 apresenta os cenários e configurações dos dois dispositivos.

Figura 11 – Cenário de aplicação dos microcontroladores

DISPOSITIVO 1 - ESP8266				
Periféricos	**Pinagem**	**Modo**	**Funcionamento**	**Widget necessário mapear**
Fita LED ARGB	Digital	OUTPUT	Ligar e desligar disparando efeito programado	Botão
LED	Digital	OUTPUT	Ligar e desligar	Botão
Sensor LDR	Analógico	INPUT	Captar intensidade de luz ambiente	Gauge ou Label
Sketch				



DISPOSITIVO 2 - ESP32				
Periféricos	**Pinagem**	**Modo**	**Funcionamento**	**Widget necessário mapear**
Fita LED RGB	Digital	OUTPUT	Controle de intensidade de cores RGB	3 Sliders
Sensor DHT22	Analógico	INPUT	Captar temperatura do ambiente	Gauge ou Label
Sketch				



Fonte: os autores

9.3. Configuração da plataforma Blynk

A partir da esquematização dos microcontroladores e definições dos periféricos e sensores que foram utilizados, foi possível configurar devidamente a plataforma. Foram adicionados dois dispositivos, cada um com suas devidas variáveis. A Figura 12 representa as configurações dos *Datastreams* que foram feitas na plataforma.

Figura 12 – Configurações dos *DataStreams* na plataforma Blynk

DISPOSITIVO 1 - ESP8266						
Id	**Nome**	**Tipo Pinagem**	**PIN**	**Valor Min**	**Valor Max**	**Tipo Valor**
1	ARGB LED Strip	Virtual PIN	V0	0	1	Integer
2	LED	Virtual PIN	V1	0	1	Integer
3	LDR Sensor	Virtual PIN	V2	0	100	Integer

DISPOSITIVO 2 - ESP32						
Id	**Nome**	**Tipo Pinagem**	**PIN**	**Valor Min**	**Valor Max**	**Tipo Valor**
1	RED LED Strip	Virtual PIN	V0	0	255	Integer
2	GREEN LED Strip	Virtual PIN	V1	0	255	Integer
3	BLUE LED Strip	Virtual PIN	V2	0	255	Integer
5	DHT Sensor	Virtual PIN	V3	-40	80	Double

Fonte: os autores

9.4. Discussão dos Resultados

Com o projeto executado, foi definido o cenário dos testes que foram realizados, alcançando os resultados esperados.

Recapitulando o cenário e suas conexões, na tela do MagicMirror² estão mapeados *widgets* para controlar e monitorar dois dispositivos, o Esp32 e o Esp8266. Através da API da plataforma Blynk, os *widgets* atualizam e recebem dados dos *Datastreams*. Do lado dos microcontroladores, são utilizadas as bibliotecas da plataforma para a comunicação entre eles. Assim, tem-se o cenário de comunicação citado nas seções anteriores, apresentado pela Figura 5.

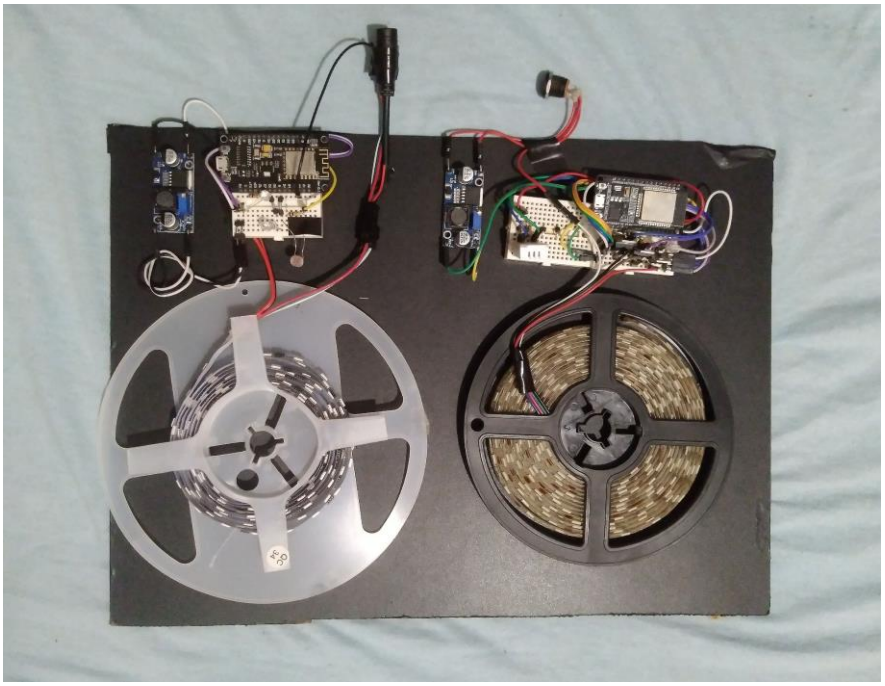
O protótipo físico do espelho inteligente é mostrado na Figura 13 apresentando o resultado do módulo instalado e configurado no MagicMirror² com as configurações apresentadas nas seções anteriores.

Do lado dos microcontroladores, o resultado da estrutura de teste é apresentado pela Figura 14.

Figura 13 – Tela do MagicMirror² com o módulo desenvolvido configurado

Fonte: os autores

Figura 14 - Estrutura de testes dos microcontroladores



Fonte: os autores

Para apresentar um pouco do cenário em funcionamento foi feito um esquema (Figura 15) mostrando a interação com alguns *widgets*, escolhendo como exemplo a parte que envolve o Esp8266, que tem como recursos configurados o *Device Status*, dois *buttons*, um *Gauge* e um *Label*. Com apenas esses recursos, foi possível testar o desempenho, em relação ao tempo de resposta, de cada categoria de *widget*: os que enviam comando para realizar ações (*Buttom* e *Slider*), os de monitoramento (*Gauge* e *Label*) e o que mostra se o dispositivo estão *online* ou *offline* (*Device Status*).

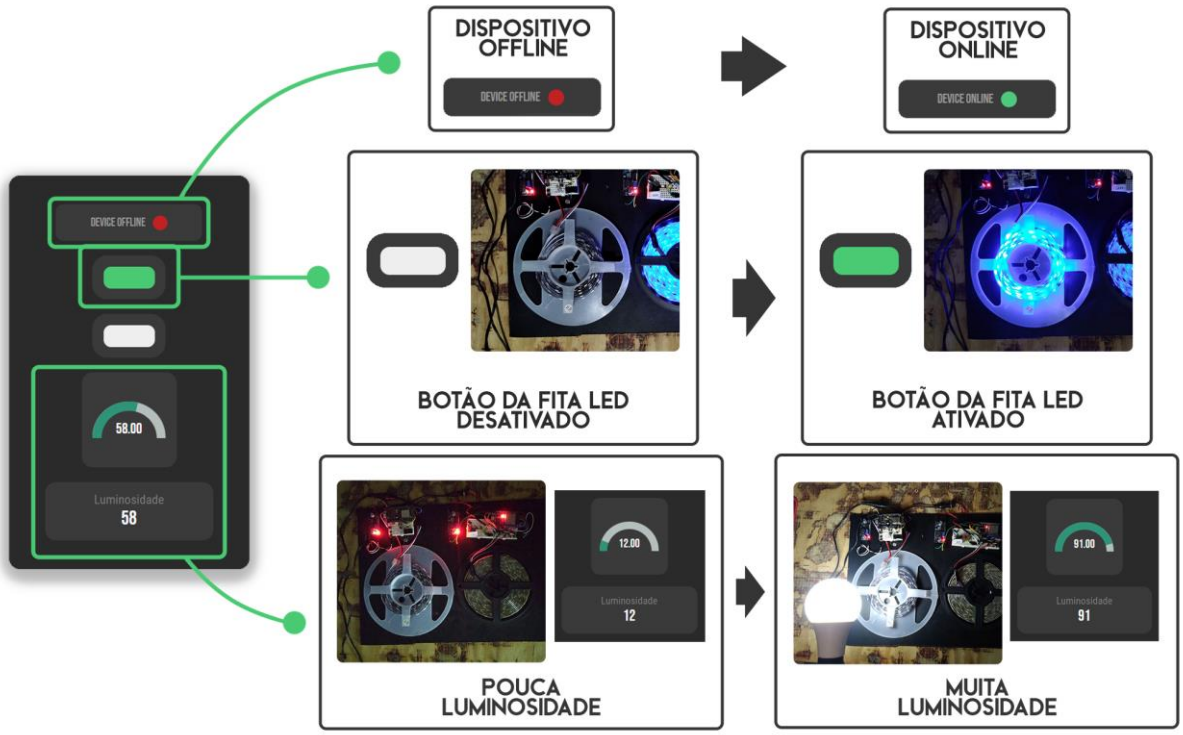
Como apresentado, o funcionamento do projeto ocorreu como esperado, porém há detalhes importantes a citar, começando pelo tempo que leva para o microcontrolador ficar *online* na tela do MagicMirror² e depois de desligado, o tempo que leva para ficar *offline*. Para o primeiro caso, o processo se resume em ligar o microcontrolador, este se conectar à internet, estabelecer comunicação com a plataforma, esta reconhecer que o dispositivo está *online* e mudar o estado, para depois ser atualizado na tela do MagicMirror² que está monitorando essa mudança de estado. Pode-se considerar que todo esse processo ocorre dentro de um tempo

satisfatório, levando apenas 8 segundos, no caso do Esp8266 e 5 segundos, no caso do Esp32. Do contrário, quando o dispositivo é desligado, leva 1:40 minutos para aparecer que o dispositivo está *offline* em ambos os dispositivos. Isso acontece por conta da própria plataforma que demora para reconhecer que o dispositivo foi desconectado.

Outro detalhe referente ao tempo de resposta é o tempo que leva para realizar a ação quando o usuário clica em um *buttom* ou interage com os *sliders* na tela do MagicMirror². Nesse caso o *delay* é praticamente nulo.

Em relação aos *widgets* de monitoramento, que são o *Gauge* e o *Label*, que ficam monitorando as mudanças de estado dos *Datastreams* com dados recebidos por um sensor, a configuração do tempo para atualizar as informações na tela é feita no código do módulo, sendo definido o intervalo de 1 segundo para realizar a próxima requisição e atualizar os valores. Na prática, esse tempo também apresenta-se consideravelmente rápido.

Figura 15 - Esquema de funcionamento do projeto



Fonte: os autores

No geral, o projeto apresenta-se com bom funcionamento e estável, ponto esse que viabiliza a publicação da primeira versão do projeto para ser utilizado por outros usuários.

9.6 Aplicabilidade em outros ambientes

Além do cenário apresentado no escopo deste artigo, é possível mencionar outros ambientes que podem ser integrados com a solução desenvolvida. Sendo a base de código do projeto composta por JavaScript e CSS, com algumas adaptações o projeto pode ser aproveitado para outros cenários, como uma tela geral de controle, extrapolando o cenário do espelho inteligente. Nesse sentido pode-se pensar em telas de controle nos mais diversos cenários, como em uma área de lazer para controlar os leds da piscina; uma sala comercial onde podem ser controlados luzes, ar condicionado, telas e projetores; e mesmo no ambiente doméstico, como uma tela de controle e monitoramento geral da casa e seus dispositivos.

Quanto à parte da lógica de controle IoT, esta é mantida pela plataforma Blynk, sendo o espelho ou tela genérica onde a solução estará executando, responsável apenas pela renderização dos controles dos dispositivos, consumo de dados da plataforma e disparada de gatilhos pelas ações do usuário. Enquanto isso, a Blynk é responsável pela comunicação com os dispositivos físicos (microcontroladores), que realizam as ações programadas.

10 Considerações finais

Com efeito, é chegada a conclusão do escopo definido para este trabalho. No decorrer do artigo foram estabelecidos os conceitos iniciais da Internet das Coisas, tratando-se especificamente do segmento de espelhos inteligentes. A partir daí buscou-se analisar mais profundamente esse cenário. Foi levantado que os *Smart Mirrors* já existem no mercado, porém ainda pouco explorados e com pouca aderência popular, com prática de valores altos e pouco acessíveis. Com o andamento da pesquisa foi possível notar então o surgimento de ideias vindas a partir da própria comunidade de desenvolvimento IoT, com soluções de código aberto, que visam facilidade de implementação e baixo custo, sendo o *framework* MagicMirror² uma dessas opções. Como proposta, foi idealizado o desenvolvimento de um novo módulo para o MagicMirror² que possibilita e facilita interações entre o espelho inteligente e microcontroladores, permitindo assim controlar e monitorar elementos da casa pela interface do espelho. Ao final da pesquisa e desenvolvimento, tem-se como resultado, um novo módulo desenvolvido para MagicMirror², utilizando a plataforma Blynk para intermediar e facilitar a comunicação entre o espelho e um microcontrolador. Como definido no escopo do projeto, foram abstraídos os cinco principais *widgets* da plataforma para controle e monitoramento*,* sendo eles o *device status*, o *buttom*, o *slider*, o *label* e o *gauge*, por meios dos quais os usuários podem monitorar informações como temperatura e luminosidade, e controlar objetos domésticos como luzes e LEDs. Após a realização dos testes com um protótipo de *Smart Mirror* e dois microcontroladores, foi possível concluir que o objetivo do projeto foi atingido, obtendo os resultados esperados

É de notável observação, o impacto que o projeto trouxe aos autores durante seu desenvolvimento, expandindo o conhecimento do universo IoT desde suas raízes e entendendo o que ele já é capaz de produzir hoje, mesmo em cenários com poucos recursos. Podem ser apontados como principais desafios a escolha das ferramentas certas para atingir o objetivo inicialmente planejado, de agregar funcionalidades IoT a um espelho inteligente, e se de fato o caminho escolhido abrangeria a proposta de valor em sua totalidade. É evidente também, que ainda há o que evoluir e ser feito. Portanto, como trabalhos futuros pretende-se o mapeamento de outros *widgets* oferecidos pela plataforma Blynk, maior personalização da interface do módulo e ainda e integração com plataformas diferentes. Esses foram os pontos analisados que

podem ser desenvolvidos no decorrer do tempo, com a difusão do módulo junto à comunidade.

O módulo para ser baixado, juntamente com sua documentação completa, pode ser acessado neste endereço: https://github.com/wTornich/MMM-BlynkCloud, do repositório do projeto no GitHub.

Referências

BHAT, Sagar. BHAT, Omkar. GOKHALE, Pradyumma. Introduction to IOT. International Advanced Research Journal in Science, Engineering and Technology, Vol.03, No.01, p. 41-44, janeiro, 2018.

CLICTEST. Evolution of Internet of Things. 2018. Disponível em: https://www.clictest.com/. Acesso em: 3 set. 2023.

COSTA, Fábio. "História do Arduino." Fábio Costa - Blog de Eletrônica e Programação. Disponível em: https://fabiocosta.net/arduino/historia-do-arduino. Acesso em: 22 maio. 2023.

CURTO CIRCUITO. Curto Circuito, © 2016-2022. Página de venda do NodeMCU V3 - ESP8266 - CP2102. Disponível em: https://curtocircuito.com.br/nodemcu-v3-esp8266-esp-12e-cp2102.html. Acesso em: 2 set. 2023.

FRYE, M. MuleSoft. What is an API? Disponível em: https://www.mulesoft.com/resources/api/what-is-an-api#:~:text=Many%20people%20ask%20themselves%2C%20%E2%80%9CWhat,data%20within%20and%20across%20organizations. Acesso em: 09 ago. 2023.

LUTKEVICH,B. TechTarget. Definition Microcontroller (MCU). Disponível em: https://www.techtarget.com/iotagenda/definition/microcontroller. Acesso em: 09.ago.2023.

MADAKAM, Somayya. RAMANWAMY, R. TRIPATHI, Siddharth. Internet of Things (IoT): A Literature Review. Journal of Computer and Communications, National Institute of Industrial Engineering (NITIE), Vol.03, No.05, p. 1-10, janeiro, 2015.

MAGIC MIRROR². Magic Mirror², © 2016. Página inicial. Disponível em: https://magicmirror.builders. Acesso em: 2 set. 2023.

NEGREIROS, Elias. Regulação e Internet das Coisas: ecossistema dos objetos inteligentes e arquitetura normativa do Direito Digital. Trabalho de Conclusão de Curso – Universidade de Brasília – UnB - Faculdade de Direito – 2018.

OLIVEIRA, Sérgio. Internet das Coisas com Raspberry Pi e Esp8266. 1ª Edição. São Paulo: Novatec Editora Ltda, Junho/2017.

OLIVEIRA, Vitor. Uso de arquitetura embarcada para automação do processo de aquisição de imagens para escaneamento 3D de baixo custo. Trabalho de Conclusão de Curso – Universidade de Brasília – UnB - Faculdade UnB Gama – FGA, 2018.

SAMORA, H. Como surgiu e qual a utilidade da IoT. 2020. Disponível em: https://www.industria40.ind.br/artigo/20246-como-surgiu-e-qual-a-utilidade-da-iot. Acesso em: 09.ago.2023.

SHARMA, Rajnish K. Net Solutions. What is a Framework in Programming?. 2023. Disponível em: https://www.netsolutions.com/insights/what-is-a-framework-in-programming/#:~:text=A%20framework%20in%20programming%20is%20a%20tool%20that%20provides%20ready,inversion%20of%20control%20(IoC). Acesso em: 02 set. 2023.

SMARTBEAR. API Endpoints - What Are They? Why Do They Matter?. Disponível em: https://smartbear.com/learn/performance-monitoring/api-endpoints/#:~:text=For%20APIs%2C%20an%20endpoint%20can,it%20will%20receive%20a%20response. Acesso em: 02. set. 2023.