APLICAÇÃO PARA CONTROLE DE FLUXO E MENSAGERIA ENTRE DISPOSITIVOS IOT

Silvio Greuel, Dalton Solano dos Reis – Orientador
Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brazil

silviog@furb.br, dalton@furb.br

Resumo: *Este artigo apresenta o processo de desenvolvimento e avaliação da utilidade de um aplicativo que tem como objetivo realizar o controle de fluxos das mensagens entre dispositivos IoT. O aplicativo foi desenvolvido utilizando o conceito de Progressive Web App e árvores de decisão. O controle de fluxo de mensageria entre dispositivos IoT possibilita ao usuário realizar configurações de execução de tarefas sem a necessidade do dispositivo estar fisicamente disponível. A aplicação Progressive Web App por sua vez, além de fornecer uma interface de gestão para os fluxos, demonstra a viabilidade de uso da tecnologia, dando acesso ao hardware, que antes somente seria possível com o desenvolvimento de uma aplicação nativa. Para validar a aplicação, foram levantados casos reais de uso agrários e industriais, além da montagem de uma bancada de homologação do fluxo e dispositivos. O aplicativo se mostrou capaz de realizar as configurações de fluxo necessários para atenderem os casos de uso e de interação com a bancada de homologação.*

Palavras-chave*: Internet das coisas. Progressive web app.*

1 INTRODUÇÃO

Um dos principais motivos para o estabelecimento dentre tantas tecnologias disponíveis na geração atual, se dá pelo modelo de código aberto, licença flexível e não proprietária em que certas tecnologias têm por base. Uma destas tecnologias se trata da própria Internet, que possui licenciamento por dito livre desde a liberação da propriedade intelectual em 1993 pelo CERN (1993, p. 2, tradução nossa) "O CERN renuncia a todos os direitos de propriedade intelectual deste código, tanto ao código fonte, quanto ao binário, e é dada permissão a qualquer pessoa para usá-lo, duplicá-lo, modificá-lo e distribuí-lo". Tal liberação permitiu a criação de muitas outras tecnologias, como: navegadores para acesso a conteúdo disponíveis via Internet, inúmeros protocolos de transmissão de dados, criptografia e até os *smartphones*. Seguindo o mesmo princípio de evolução dos *smartphones*, espera-se também, que de maneira semelhante, a utilização de dispositivos Internet of Things (IoT) cresça e se estabeleça. A procura por soluções domésticas abriu caminho para a automação residencial, por meio de sistemas que se propõem a melhorar a qualidade de vida e economizar recursos (SOUZA, 2016). A fragmentação de softwares e protocolos dificulta a comunicação de dispositivos IoT e é neste ponto que o *open source* se torna importante. Por mais que cada fabricante programe seus produtos para funcionar de forma especifica, uma vez que eles estão em uma plataforma de código aberto, é possível fazer com que todos esses dispositivos diferentes trabalhem juntos (OLHAR DIGITAL, 2016).

Atualmente, existem diferentes tipos de serviços e aplicativos que tem como principal objetivo realizar o controle e gestão de dispositivos IoT. Dentre estes aplicativos, se destaca a aplicação HomeKit (APPLE, 2018), que disponibiliza um framework para controle de alguns dispositivos, porém, somente dispositivos proprietários ou com o certificado exigido pela marca.

APPLE (2018) indica que:

> [...] Com o app Casa, você pode acessar remotamente todos os acessórios inteligentes pela Apple TV ou iPad. É possível fechar o portão da garagem, ver a câmera de vídeo da porta da frente, pedir à Siri (assistente virtual) para diminuir a temperatura ou qualquer outra coisa que você já está acostumado a fazer quando usa o app Casa em casa. (APPLE, 2018, p. 2, tradução nossa).

Outra aplicação que merece destaque, não só por se conectar com dispositivos IoT, mas, pelo modelo de controle de fluxos e mensageria entre variados serviços, chama-se If This Then That (IFTTT). Vorapojpisut (2015, p. 2, tradução nossa) "O aplicativo IFTTT permite aos usuários criarem, customizarem e habilitarem correntes condicionais, quais são chamadas de receitas, em que são ativadas com base em alterações em outros serviços, como Facebook, Twitter e Youtube".

Com a simples apresentação das soluções HomeKit e IFTTT, é possível notar a comodidade que os aplicativos de automação residencial, que fazem a gestão e o controle de dispositivos IoT, serviços sociais (facebook, twitter) e serviços privados (posicionamento geográfico), trazem para o cotidiano de uma pessoa, como afirmado por (FLORES; LUNDMARK; MÄHR, 2005, p. 3, tradução nossa) "Conveniência é um dos principais pontos de venda para automação residencial [...]". Apesar dos benefícios do uso de tais serviços estarem em evidência, a exemplo da aplicação HomeKit, percebe-se, que manter exclusividade em seus serviços, limita sua utilização. Expondo então, a falta de aplicativos multiplataforma, serviços e frameworks com uma licença não proprietária, de código livre e alteração permissiva.

O presente estudo, foca no desenvolvimento de uma aplicação multiplataforma, de código aberto possibilitando alteração, para controle de dispositivos IoT, em que terá como referência o modelo de controle de fluxo e mensageria similar a aplicação IFTTT. Tendo em vista o desenvolvimento multiplataforma, será utilizada a tecnologia Progressive Web App (PWA), que se trata de uma aplicação Web a qual são concedidas algumas permissões de acesso no contexto do sistema operacional, antes somente concedidas para aplicações nativas, como explica Fransson e Driaguine (2017, p. 5, tradução nossa) "Uma aplicação PWA é uma aplicação Web, que é incrementada com algumas tecnologias que permitem um comportamento similar a uma aplicação nativa em dispositivos móveis, enquanto mantém também seu funcionamento em um navegador".

Este trabalho tem como objetivos específicos: explorar integração de instalação nativa no sistema operacional, validando o acesso off-line a aplicação; explorar o uso da aplicação utilizando uma bancada de homologação, descrevendo de modo que possa ser futuramente reproduzido; levantar casos de uso no ambiente agrário e apresentar uma solução aplicável utilizando o aplicativo produzido por este trabalho; levantar casos de uso no ambiente industrial e apresentar uma solução aplicável utilizando o aplicativo produzido por este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os aspectos teóricos utilizados para o desenvolvimento deste projeto, primeiramente dando uma breve descrição dos conceitos da IoT e de aplicações PWA, em seguida, apresentando os trabalhos correlatos que serviram como referência no desenvolvimento deste trabalho.

2.1 CONCEITOS

Nesta seção serão apresentados os dois principais assuntos que fundamentam o desenvolvimento da aplicação: a IoT, que se refere ao conceito dos dispositivos interligados pela Internet e a PWA, a qual permite que uma página da Web possa ser utilizada como uma aplicação nativa do sistema operacional.

2.1.1 Internet das coisas

Grande parte dos dispositivos ao nosso redor hoje, suportam alguma forma de computação e meio de comunicação. Esses dispositivos, como celulares, sensores e Laptops fazem parte do dia a dia (ATHREYA; TAGUE, 2013, p. 25). A função atual desses dispositivos é a interação uns com os outros, através de uma rede como a Internet, essa interação dos dispositivos permite que eles atinjam objetivos comuns (ATHREYA; TAGUE, 2013, p. 25). Tal ecossistema ou paradigma é chamado Internet das Coisas (IoT), os dispositivos sendo referidos como as coisas, capazes de realizar processamento, coletar informações e distribuir essas informações para outros dispositivos ou servidores de análise e processamento.

Bari, Mani e Berkovich (2013, p. 48) relevam que o termo IoT pode ser mais apropriadamente referido como a Internet que relacionam as coisas, mas em que as coisas, são na verdade, informações sobre coisas, descrevendo-as como metadata. Para aproximar o mundo físico e o mundo das informações, sensores coletam dados do ambiente, fornecendo informações necessárias, que após o processamento, são traduzidas em respostas físicas pelos atuadores (BARI; MANI; BERKOVICH, 2013, p. 48). Coetzee e Eksteen (2011, p. 1) descrevem IoT como uma visão em que os objetos se tornam parte da Internet, onde os objetos são identificados de maneira única, possuem posição, *status* e são acessíveis via rede, por fim, tornam possível a adição de serviços e inteligência, realizando a expansão destes objetos pela Internet, fundindo o mundo digital e físico, impactando nos ambientes profissional, pessoal e social.

Tornar os dados a base para automação e controle, significa converter os dados e as análises coletadas através da IoT, em instruções, que são enviadas para atuadores que, por sua vez, modificam os processos (CHUI; LÖFFLER; ROBERTS, 2010). Chui, Löffler e Roberts (2010) completam que o ciclo da coleta de dados e resposta dos dados em automação de tarefas, podem aumentar a produtividade, pois

estes sistemas se ajustam automaticamente a situações complexas, tornando desnecessárias as intervenções humanas. Sobre informação e análise de dados, Coetzee e Eksteen (2011, p. 4) consideram que a IoT faz com que os serviços de tomada de decisão sejam mais eficientes, pois recebem dados precisos com mais frequência, permitindo uma análise mais acurada do atual *status quo* do ambiente em que os objetos IoT se encontram. A coleta de informação e análise aplica-se por exemplo, ao rastreamento de produtos em uma cadeia de logística ou a consciência situacional, por exemplo, sensores que coletam condições ambientais, como temperatura e umidade (COETZEE; EKSTEEN, 2011, p. 4).

A IoT pode ser usada para melhorar a vida das pessoas, no âmbito pessoal ou profissional, com o objetivo da interação inteligente entre objetos para que eles realizem ações de forma onde as características eficiência e agilidade sejam garantidas com o objetivo de ajudar os usuários. As aplicações de IoT surgiram em muitas áreas, como logística, transporte, ambientes inteligentes (casas, prédios, infraestrutura), energia, defesa e agricultura. Em essência, a IoT tem a capacidade de influenciar significativamente em vários atributos da sociedade. Os principais desafios relacionados a IoT, estão relacionados aos mecanismos de conectividade, já que todos os objetos precisam estar dispostos em uma rede, contendo um endereçamento único. Tal endereçamento dá a possibilidade dos dispositivos IoT se comunicarem, tanto no fornecimento de dados, quanto no controle de atuação física ou digital. Outro desafio, diz respeito aos mecanismos de processamento de dados em tomadas de decisão. Se faz necessário a tradução dos dados em um modelo dinâmico e interligado, onde ocorridas as entradas de inúmeros dados, possibilite a realização de ações físicas no ambiente.

2.1.2 Progressive WEB APP (PWA)

Uma PWA, é essencialmente um conceito que determina algumas práticas na criação de um aplicativo acessível via navegador, que é progressivamente aprimorado com as evoluções tecnológicas da Web. PWAs são inicialmente veiculados em um servidor Web, semelhante a aplicativos Web comuns, mas que podem ser instalados nativamente nos dispositivos, quando visitados através de navegadores compatíveis. O conceito PWA, segundo Kapoor (2018) não se trata de uma nova tecnologia, mas sim de um conjunto de práticas recomendadas, em função de transformar similar a experiência de um aplicativo Web, para com a experiência de um aplicativo nativo.

Segundo Lepage (2018) as PWAs possuem algumas características fundamentais, sendo elas: progressiva, funciona para todos os usuários independentemente da escolha do navegador; responsiva, são executadas independentemente do tipo do dispositivo; independência da rede, não se faz necessário estar conectado à Internet; seguro, somente possível ser servido via uma conexão segura com a Internet; detectável, é reconhecido pelo dispositivo como uma aplicação nativa; instalável, o dispositivo permite a instalação do aplicativo na área de trabalho do sistema operacional; compartilhável, o aplicativo pode ser compartilhado por meio de um endereço eletrônico. Aplicativos Web, devem atender três requisitos para que sejam reconhecidos como um aplicativo PWA: devem ser servidos via uma conexão segura com a Internet, no caso HTTPS; devem conter um arquivo de configuração chamado de `app manifest`, que tem a funcionalidade de atribuir características nativas ao aplicativo; devem ao menos, registrarem um `service worker`, responsável por permitir sua funcionalidade independente da conexão ou não com a Internet.

O `app manifest` segundo a W3C (2018) ainda está sendo definido e trata da especificação de um documento JSON contendo configurações do aplicativo Web. As configurações possíveis incluem, entre outros aspectos, adicionar o nome do aplicativo, atribuir ícones para o uso na área de trabalho e definir a página inicial a ser aberta quando o aplicativo for executado (W3C, 2018). O `app manifest` também permite que os desenvolvedores declarem uma orientação padrão para o aplicativo, além de fornecer a capacidade de indicar modo de exibição para o aplicativo, como a cor da barra de tarefas ou se o aplicativo deve ser executado em tela cheia (W3C, 2018).

Segundo a Mozilla D (2018) um problema primordial que os usuários da Web sofreram por anos, é a perda de conectividade com a Internet, pois a conectividade com a Internet é essencial para que haja o download do aplicativo Web. Este problema ainda persiste, porém existe a possibilidade do uso dos `services workers` como atores na mitigação deste problema. Os `service workers` atuam essencialmente como servidores proxy que permanecem entre o navegador e a Internet (MOZILLA B, 2018). Destinam-se, entre outras coisas, a permitir a criação de aplicativos onde não se tenha a necessidade de conectividade com a Internet, interceptando solicitações de rede e tomando as medidas adequadas com base na disponibilidade da rede (MOZILLA B, 2018). Além das funcionalidades atuantes sobre a disponibilidade da Internet, os `service workers` permitem o acesso a funcionalidades nativas como notificação e sincronização em segundo plano (MOZILLA B, 2018).

Um `service worker` possui um ciclo de vida completamente separado do ciclo de vida de um aplicativo Web (LEPAGE, 2018). Para instalar um `service worker` em um aplicativo Web, é preciso registrá-lo (LEPAGE, 2018). O registro de um `service worker` fará com que o navegador inicie a etapa de instalação em segundo plano (LEPAGE, 2018). Durante a etapa de instalação, alguns arquivos podem ser escolhidos de forma a estarem disponíveis sem a necessidade de uma futura conexão com a Internet (LEPAGE, 2018). Se todos os arquivos forem armazenados com êxito, o `service worker` será instalado, se algum dos arquivos não forem baixados e armazenados, a etapa de instalação falhará e o responsável pelo `service worker` não será ativado, ou seja, o aplicativo Web não será considerado um aplicativo PWA (LEPAGE, 2018). Quando instalado, a etapa de ativação do `service worker` será seguida, esta etapa permite a eliminação de arquivos armazenados, que não são desejados e após a etapa de ativação, o `service worker` estará ativo em todas as páginas definidas no `app manifest` (LEPAGE, 2018).

2.2 TRABALHOS CORRELATOS

Alguns trabalhos, nos quais as características se assemelham ao proposto, serão listados neste tópico. O primeiro, trata da definição da estrutura de execução de um aplicativo similar ao modelo da aplicação IFTTT, elaborado por Vorapojpisut (2015) e apresentado no Quadro 1. O segundo trabalho, realiza um comparativo entre variados protocolos de comunicação com foco em dispositivos IoT foi elaborado pela Adlink (2017) e apresentado no Quadro 2. O terceiro trabalho, coloca em prova, a viabilidade da utilização de uma aplicação utilizando a tecnologia PWA defronte a uma aplicação nativa, elaborado por Fransson e Driaguine (2017) e apresentado no Quadro 3.

Vorapojpisut (2015), descreve um modelo de aplicação definindo as funcionalidades e conceitos básicos que compõem o serviço IFTTT, segundo Vorapojpisut (2015, p. 2) a abordagem proposta pelo serviço IFTTT oferece vantagens entre funcionalidade e usabilidade. Em comparação com a aplicação desenvolvida neste trabalho, também será aplicado o modelo de gatilhos e ações do autor. O modelo de gatilhos e ações, define duas principais aplicações. A aplicação de gestão e controle, realizando as funções básicas do aplicativo, como cadastro de usuários, serviços de terceiros, conexões entre os serviços e fluxos. A aplicação de processamento, responsável por receber eventos de execução de fluxos, realizando a reconstrução do fluxo configurado pelo usuário, delegando os dados da mensagem recebida para o fluxo e por fim, dando o início da execução do fluxo, que perante a configuração do usuário, irá executar os procedimentos definidos.

Quadro 1 – Trabalho Correlato 1

Referência	Vorapojpisut (2015).
Objetivos	O conceito principal é formular uma automação mínima que integre múltiplas ações em fluxos de decisão que o usuário escolhe. O trabalho propõe uma estrutura de software que promova a flexibilidade e a capacidade de manutenção e outras métricas de qualidade.
Principais funcionalidades	Definição de um sistema de execução de ações arbitrarias em fluxos registrados pelo usuário.
Ferramentas de desenvolvimento	Além de textos que definem o sistema, o autor usou também a modelagem UML para apresentar sua solução.
Resultados e conclusões	O autor obteve uma modelagem simples para o problema, em que somente duas aplicações são necessárias, uma aplicação de controle e outra de processamento. A aplicação de controle tem como principal funcionalidade realizar a gestão de usuários, ações e fluxos. Aplicação de processamento recebe mensagens, identificando qual fluxo deve ser executado para então iniciar o processo.

Fonte: elaborado pelo autor.

Quadro 2 – Trabalho Correlato 2

Referência	Adlink (2017).
Objetivos	O trabalho tem como principal objetivo, comparar os protocolos de comunicação DDS, AMQP, MQTT, JMS, REST, CoAP, e XMPP, tendo como principal foco sua utilização em soluções IoT.
Principais	Apresentação e descrição dos tópicos de análise, sendo eles: segurança,

funcionalidades	performance, qualidade de serviço, intratabilidade, estratégia de conexão e estratégia de dados.
Ferramentas de desenvolvimento	Método científico e analítico, realizando pesquisa sobre as especificações dos protocolos e os comparando em tópicos comuns.
Resultados e conclusões	Como resultado, se obteve uma tabela comparativa de conformidades entre os protocolos.

Fonte: elaborado pelo autor.

Quadro 3 – Trabalho Correlato 3

Referência	Fransson e Driaguine (2017).
Objetivos	Validar o uso de PWAs como substituto para aplicações nativas.
Principais funcionalidades	Apresentação de duas aplicações, uma aplicação PWA e uma aplicação nativa, as funcionalidades das duas aplicações consistem, em acesso a câmera e uma tela para realizar o teste de performance de renderização da aplicação.
Ferramentas de desenvolvimento	Metodologia cientifica para identificar a performance das aplicações PWA em comparação com aplicações nativas, analisando o tempo de acesso, a câmera e o tempo de renderização da aplicação. Metodologia de viabilidade, listando as possibilidades de acesso de uma PWA ao sistema nativo.
Resultados e conclusões	Criação de uma tabela contendo a comparação direta de performance entre uma aplicação PWA e uma aplicação nativa, de modo geral, ao se tratar do tempo de acesso a câmera, as duas aplicações não se diferem em muito, já, o tempo de renderização da aplicação PWA em comparação com a aplicação nativa, deixa a desejar. Criação de uma tabela identificando as possibilidades de acesso de uma aplicação PWA, indicando que possibilidade de uso do *bluetooth*, geolocalização, sistema de arquivos, modo off-line entre outros.

Fonte: elaborado pelo autor.

O trabalho desenvolvido pelo grupo ADLINK (2017), foca nas tecnologias de mensageria que estão emergindo como as mais importantes para os dispositivos IoT, como afirmado pelo grupo ADLINK (2017, p. 5, tradução nossa) “Existem muitas tecnologias de mensagens diferentes. No entanto, um subconjunto muito menor está emergindo como o mais importante”. São analisados os protocolos, *Data Distribution Service for Real-Time Systems* (DDS), *Advanced Message Queuing Protocol* (AMQP), *MQ Telemetry Transport* (MQTT), *Java Message Service* (JMS), *Representational State Transfer* (REST), *Constrained Application Protocol* (CoAP), *eXtensible Messaging e Presence Protocol* (XMPP). A conclusão obtida pelo trabalho, indica como utilização ideal para dispositivos IoT, os protocolos MQTT, AMQP e REST. Realizando a análise das conclusões obtidas pelo trabalho correlato apresentado no Quadro 2, houve a escolha do uso de dois protocolos. As propriedades de qualidade de serviço definidas pelo protocolo AMQP, exercem vantagem sobre as demais ao se tratar da garantia de entrega, sendo então a escolha de protocolo para o processamento dos fluxos. Para a comunicação dos dispositivos IoT com o aplicativo, o protocolo MQTT se torna viável por ter como um dos principais atributos, a baixa latência no envio das mensagens, propriedade importante, quando consideramos a baixa eficiência das conexões com a Internet, como intermitência da conexão ou baixa velocidade de transmissão dos dados.

O trabalho de Fransson e Driaguine (2017) apresenta inúmeras análises comparando aplicações utilizando a tecnologia PWA e aplicações nativas. Dentre as análises, pode se destacar, o consumo de bateria, o tempo de renderização completa e acessibilidade das funções disponibilizadas pelo sistema móvel, por exemplo *Global Positionig System* (GPS). A aplicação desenvolvida neste trabalho irá utilizar também o conceito de PWA fazendo o uso de acesso nativo ao *Bluetooth* e *Near Field Comunication* (NFC).

3 DESCRIÇÃO DA APLICAÇÃO

Este capítulo pretende apresentar os detalhes de especificação e implementação do aplicativo. Para tanto, são apresentadas quatro seções. A primeira seção apresenta a visão geral do aplicativo, visando a ambientar o leitor quanto as funcionalidades do fluxo de comunicação. A segunda seção

apresenta a arquitetura do aplicativo, detalhando o funcionamento das aplicações de gestão e processamento. A terceira seção apresenta os dispositivos utilizados para realizar a homologação das funcionalidades da aplicação. Por fim, a quarta seção apresenta os protocolos de comunicação utilizados, descrevendo sua utilidade na estrutura de mensagens.

3.1 VISÃO GERAL

O aplicativo tem por objetivo a automação e intercomunicação entre dispositivos IoT, para que seu utilizador tenha o total controle dos fluxos e tomadas de decisões, mesmo estando remoto. Os principais requisitos para a criação do sistema são: facilidade de configuração, utilização de hardware de baixo custo, utilização de tecnologias abertas, configuração remota dos dispositivos, sem a necessidade de atuação física por parte do utilizador, arquitetura flexível e extensível. O aplicativo é capaz de se comunicar com dispositivos via protocolo MQTT, de modo que é possível coletar informações como umidade e temperatura e executar ações. Por exemplo: é possível controlar o acionamento de um motor quando o sensor de temperatura indicar uma temperatura abaixo do que informado pelo usuário.

Para realizar a comunicação entre os dispositivos, foi criado um protocolo de mensagem. O protocolo define a estrutura de campos, contendo tipo, nome e valor. O protocolo define que mensagens podem possuir um ou mais campos, cada campo, deverá ter o formato `tipo:descrição:valor`, caso haja mais campos na mensagem, deverá ser adicionado o caractere de final de linha `CR` para serem diferenciados. Os tipos possíveis para os campos são: `guid`, `bool`, `int`, `float`, `double` e `string`. O protocolo garante a tolerância de duas falhas, a falha de tipo não reconhecido e a falha de campo não informado em um formato correto. Caso seja informado um tipo não reconhecido, o protocolo considera como tipo padrão, o tipo `string`. Caso o campo não esteja em um formato correto, o protocolo ignora o campo. O Quadro 4 demonstra a possibilidade de interoperabilidade dos tipos definidos pelo protocolo e as linguagens de programação JAVA, CSHARP, C++ e GO.

Quadro 4 – Mapa de interoperabilidade dos tipos

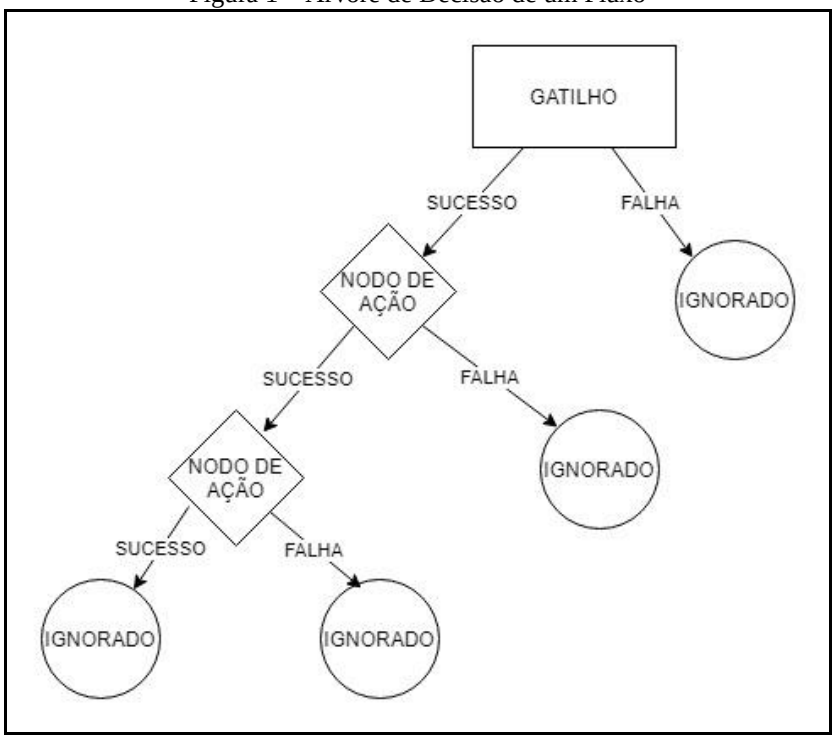
PROTOCOLO	**JAVA**	**CSHARP**	**C++**	**GO**
Guid	UUID	guid	N/A	N/A
Bool	boolean	bool	bool	bool
Int	int	int	int32_t	int32
float	float	float	float	float32
double	double	double	double	float64
string	string	string	std::string	string

Fonte: elaborado pelo autor.

Para realizar a execução do processamento das ações, foi definida uma entidade chamada de `fluxo`. A elaboração do `fluxo` como apresentada na Figura 1, foi concebida em uma estrutura de dados conhecida como árvore de decisão, que é uma estrutura muito similar a de uma árvore binária, porém se difere na capacidade de um nodo realizar decisão de caminho. Em definição a árvore binária é composta por nodos possuindo o mesmo algoritmo de decisão, já a árvore de decisão permite que os nodos possuam diferentes algoritmos. Outro aspecto da árvore de decisão utilizada pelo fluxo, é a criação de um contexto pelo primeiro nodo da árvore e, o repasse deste contexto, para a execução do próximo nodo.

A Figura 1 apresenta uma árvore de decisão utilizada pelo fluxo, nota-se que o primeiro nodo possui o nome de `gatilho`, já os demais `nodos de ação`, possuem nodos adjacentes, representando a sequência de `sucesso` ou `falha` da execução do nodo.

Figura 1 – Árvore de Decisão de um Fluxo



Fonte: elaborado pelo autor.

O `gatilho` é o primeiro nodo da árvore de decisão, são duas as responsabilidades do gatilho, a primeira é validar se a execução da árvore de decisão é necessária, e a segunda é realizar a criação de um contexto de execução, o qual será repassado para os próximos nodos. A aplicação define dois principais gatilhos, o gatilho `MQTT`, que somente é disparado caso uma mensagem MQTT seja processada pelo processador de fluxos e o gatilho `BUTTON`, que somente é disparado por ação explicita do usuário.

O `contexto` é uma estrutura de dados chamada de dicionário. Tal estrutura é utilizada em um modelo de chave e valor, sendo a chave um texto qualquer e o valor um valor dinâmico qualquer, ou seja, a estrutura permite que inúmeros campos sejam atribuídos a indiferentes tipos. O contexto é fundamental para realizar o repasse de informações entre os nodos da árvore de decisão, fazendo com que dados gerados pelos nodos, sejam salvos e repassados para os nodos subsequentes.

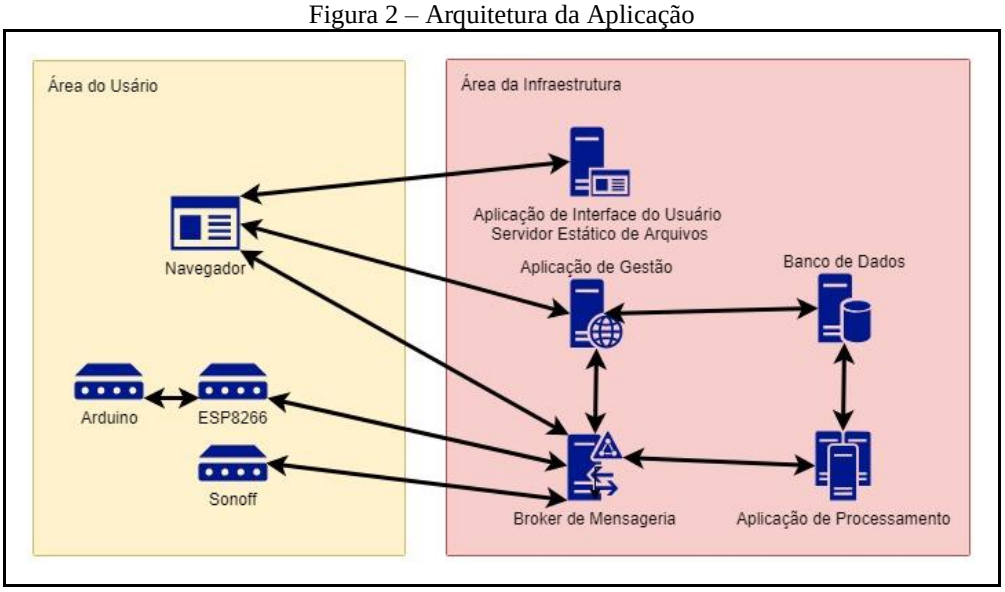
Os `nodos de ação` são encarregados de realizar execuções arbitrárias, podendo ou não, acessar e alterar o contexto de execução, que foi repassado pelo nodo pai. Após a execução da ação, o nodo é responsável por realizar a chamada de um dos dois nodos filhos, sendo eles o nodo de sucesso, quando a execução for concluída conforme o esperado, falha, quando houver falha na execução, porém, quando ocorrer alguma falha não esperada, todas as execuções subsequentes serão ignoradas, fazendo que a execução do `fluxo` termine.

A aplicação define sete ações distintas, sendo eles `HTTP POST`, `HTTP GET`, `ATTRIBUTE`, `MQTT PUBLISH`, `DECISION`, `GPIO` e por fim `GUARD`. A ação `HTTP POST`, realiza o envio de uma requisição `HTTP` utilizando o verbo POST, contendo no corpo da mensagem, um campo disponível no contexto da aplicação que foi selecionado pelo usuário. A ação `HTTP GET`, realiza o envio de uma requisição `HTTP` utilizando o verbo GET, a resposta da mensagem é salva no contexto da aplicação em um campo selecionado pelo usuário. A ação `ATTRIBUTE`, adiciona um campo no contexto da execução, dado um tipo, dado um valor e dado um nome selecionado pelo usuário. A ação `MQTT PUBLISH` realiza a publicação de uma mensagem utilizando o protocolo `MQTT`, para o tópico do dispositivo selecionado pelo usuário utilizando como corpo da mensagem, um campo selecionado pelo usuário. A ação `DECISION` executa uma validação usando uma das expressões `Equal`, `NotEqual`, `GreaterThan`, `GreaterThanOrEqual`, `LessThan` e `LessThanOrEqual`, entre dois valores contidos no contexto da execução, selecionados pelo usuário. A ação `GPIO`, realiza a publicação de uma mensagem utilizando o protocolo MQTT, para um dispositivo selecionado pelo usuário, tal mensagem possui o identificador da `gpio` e o seu *status* desejado, dentre os *status* o usuário tem a possibilidade de escolha entre `High`, `Low` e `Toggle` (qual alterna entre `High` e `Low`). A ação `GUARD`, valida a existência de um campo no contexto da execução, caso o campo não esteja presente no contexto é executada a ação de falha.

3.2 ARQUITETURA

Durante este sub tópico, será exposta a funcionalidade dos componentes pertencentes a arquitetura da aplicação. A arquitetura geral é dividida em duas áreas, a área do usuário, composta por tudo que o usuário final do aplicativo consegue ver/interagir e, a área da infraestrutura, composta pelo aplicativo desenvolvido neste trabalho. A área do usuário é representada pelos dispositivos IoT e pela interface de gestão do aplicativo, já a área da infraestrutura, é representada pelo *broker* de mensageria, pela aplicação de gestão, pela aplicação de processamento e pelo banco de dados. Ambas as áreas são retratadas na Figura 2, que apresenta também, as conexões de comunicação entre os componentes.

Figura 2 – Arquitetura da Aplicação



Fonte: elaborado pelo autor.

A aplicação de gestão é realizada via *Application Program Interface* (API), uma interface de controle e gestão. Os principais recursos que API disponibiliza são, controle de conta, gestão de dispositivos, gestão de fluxos e por fim, listagem e acionamento de botões. O controle de conta permite o cadastro de um novo usuário ou autenticação de um usuário já existente. Ao realizar a autenticação de um usuário já existente, a API retorna um JSON Web Token (JWT) que deverá ser utilizado nas demais requisições efetuadas para a API. A gestão dos dispositivos, permite efetuar a listagem dos dispositivos, a remoção de um dispositivo e o cadastro de um novo dispositivo. A gestão de fluxos permite a listagem dos fluxos existentes, a remoção de um fluxo, a adição de um gatilho no fluxo e a adição dos demais nodos de ação. A listagem de botões lista todos os fluxos registrados em que, o gatilho do fluxo, é um gatilho do tipo botão, também permite a execução do gatilho do tipo botão.

Se fez o uso do conceito apresentada na seção 2.1.2 para criar aplicação de interface do usuário. A aplicação realiza o consumo da API da aplicação de gestão, apresentando os recursos em formulários de preenchimento. Há a possibilidade de realizar uma conexão via WebSocket diretamente com o *broker* de mensageria, dado que a conexão com o *broker* esteja ativa, o dispositivo passará a ser reconhecido pelo *broker*, podendo então receber mensagens diretas via protocolo MQTT. A aplicação de interface do usuário, além do acesso a API de gestão e ao *broker* de mensageria, traz a possibilidade de acesso a APIs nativas do dispositivo. As APIs são: Web *Bluetooth*, que realiza pareamento com dispositivos BLE; NFC (*Near Field Communication*), que realiza escrita e leitura de RFIDs (*Radio-Frequency IDentification*); *Notification*, que permite o envio de mensagens de notificação para o sistema operacional; *Vibration*, faz que o dispositivo vibre, somente disponível para os dispositivos móveis.

Se utilizou o *broker* de mensageria RabbitMQ, dentro da aplicação, o *broker* mantém três principais funções. A primeira função realizada pelo *broker* é ser o centralizador de todas as mensagens enviadas via protocolos MQTT, AMQP e WebSocket. Segunda função realizada pelo *broker* é fazer com que todas as mensagens sejam enviadas para a aplicação de processamento. Terceira e última função realizada pelo *broker* é permitir interoperabilidade entre os protocolos MQTT, AMQP e WebSocket.

A aplicação de processamento, realiza o processamento de todas as mensagens enviadas para o *broker* de mensageria, caso uma das mensagens obedeça aos critérios de ativação do gatilho do fluxo, o processador cria um contexto de execução, preenchendo-o com os dados obtidos da mensagem, por fim, inicia a execução do fluxo passando o contexto então criado.

3.3 DISPOSITIVOS

Este sub tópico apresenta os dispositivos utilizados para realizar a homologação da aplicação, para isso se adquiriu os dispositivos ESP8266, Arduino Uno, Sonoff e *Bluetooth* HM-10. Estes dispositivos agem como sensores e atuadores e somente se comunicam via definições dos fluxos.

O ESP8266 é um dos dispositivos utilizados, segundo Oliveira (2017, p. 3) "O ESP8266 é um microcontrolador produzido pela empresa Espressif Systems. Esse micro controlador possui um sistema de comunicação WiFi próprio, que é o seu grande diferencial", como o dispositivo tem uma conexão WiFi própria e pode ser interligado com outros dispositivos Oliveira (2017, p. 3) afirma que o ESP8266 "é largamente utilizado como módulo WiFi para outros micros controladores, como o Arduino, por exemplo, apesar de possuir um processador próprio". O Arduino é uma plataforma eletrônica de código aberto que possui seus próprios micro controladores e a sua própria IDE. Oliveira (2017, p. 8) afirma que "O Arduino tem uma grande comunidade de usuários, diversos trabalhos e projetos que se utilizam dessa plataforma, pela sua facilidade de uso, eficiência e sua IDE gratuita, além do grande número de ferramentas já implementadas para o mesmo"

O Sonoff é baseado no ESP8266 e o mesmo possui um *firmware* que é capaz de fazer com que o módulo se conecte ao servidor na nuvem por meio de uma rede WiFi, desta forma permite que todo o controle sobre a placa possa ser feito remotamente (MASTERWALKERSHOP, 2018). O HM-10 é um dispositivo *Bluetooth* 4.0 disponível com base no sistema *Bluetooth Low Energy* (BLE) (MIT, 2015). O BLE funciona de uma maneira muito diferente do *Bluetooth* convencional. O BLE é projetado para aplicações de baixa energia, usando pequenos pacotes de dados enviados com pouca frequência (MIT, 2015).

3.4 PROTOCOLOS DE TRANSPORTE

Os protocolos de transporte permitem que mensagens sejam entregues entre dispositivos. Um fato importante na escolha destes protocolos, é a possibilidade de tradução dos protocolos. Este capítulo identifica os protocolos utilizados e suas funções dentro da aplicação. O protocolo AMQP, atende o fato de não ser um protocolo proprietário, nem limitado quando se trata do aspecto da possibilidade de uso entre diferentes aplicações como explica ADLINK (2017, p. 6, tradução nossa) "AMQP é um protocolo que tem o foco principal em transmitir mensagens [...], em uma solução de uso comum e não proprietário permitindo interoperabilidade". Este protocolo possui algumas simples características, como, o fato do transporte da mensagem ser em formato binário, acarretando em rapidez e, o fato do protocolo ser acessível sem abstração diretamente pela aplicação, adicionando flexibilidade e interoperabilidade em seu uso, como descrito por ADLINK (2017, p. 6, tradução nossa) "AMQP foi desenhado para suportar padrões de variados protocolos de comunicação existentes". Outros atrativos do protocolo, seriam o fluxo da transmissão das mensagens em um modelo de produtor/consumidor e os tipos de garantia de entrega disponíveis, conforme ADLINK (2017, p. 6, tradução nossa) "[...] no máximo uma vez (mensagem pode ou não ser entregue, porém, somente uma vez), ao mínimo uma vez (mensagem é entregue ao menos uma vez, porém, pode ser entregue mais de uma vez) e exatamente uma vez (mensagem é entregue uma única vez) ".

O protocolo AMQP é utilizado como protocolo base de comunicação entre as aplicações da infraestrutura. A aplicação de processamento, consome as mensagens, realiza o processamento do fluxo, caso seja definido no fluxo, também realiza a publicação de uma mensagem. O protocolo de publicação de mensagens de assinatura, foi aprovado como um padrão ISO em 2015. É um protocolo muito leve e flexível, requer apenas 2 bytes de informação de cabeçalho. O MQTT é uma arquitetura baseada em *broker*. O *broker* consiste em uma lista de todos os tópicos atualmente disponíveis. Todas as mensagens precisam passar pelo *broker*. O *broker* mais conhecido é chamado de Mosquitto, mas para a aplicação deste trabalho, foi utilizado o *broker* RabbitMQ. Um sistema MQTT típico consiste em um único ou vários *broker*s. O protocolo MQTT foi desenhado para ser utilizado em locais em que há alta latência, ou seja, o envio de mensagens ponta a ponta pela Internet é demorada, conforme destaca ADLINK (2017, p. 6, tradução nossa) "MQTT foi desenhado para transmitir mensagens em redes limitadas por velocidade. "

Protocolo MQTT foi utilizado neste trabalho, como protocolo de comunicação entre dispositivos IoT e o *broker* de mensageria. Ao chegar no *broker* de mensageria, as mensagens utilizando o protocolo MQTT são traduzidos para o protocolo base AMQP, permitindo o consumo das mensagens pela aplicação de processamento. A tradução inversa do protocolo AMQP para MQTT também ocorre, quando a aplicação de processamento necessita realizar o envio de uma mensagem para os dispositivos IoT.

Hypertext Transfer Protocol (HTTP) é um protocolo de camada de aplicação para transmitir documentos hipermídia, como HTML. Ele foi projetado para comunicação entre navegadores da Web e

servidores da Web, mas também pode ser usado para outras finalidades (MOZILLA A, 2018). O HTTP segue um modelo clássico de cliente-servidor, com um cliente abrindo uma conexão para fazer uma solicitação e aguardando até receber uma resposta (MOZILLA A, 2018). Por ser um protocolo unidirecional, não há a possibilidade de realizar a tradução entre o HTTP e os protocolos AMQP, MQTT e MQTT sobre WebSocket, estre protocolo então, não é utilizado para realizar comunicação entre os dispositivos e o *broker* de mensageria, mas para fazer uso do protocolo para realizar a comunicação da aplicação de interface do usuário e a aplicação de gestão.

O WebSocket é uma tecnologia avançada, que possibilita a abertura de uma seção de comunicação interativa bidirecional entre o navegador do usuário e um servidor, dando a possibilidade de envio de mensagens para o servidor (MOZILLA C, 2018). Como o WebSocket dá a habilidade de comunicação bidirecional com o servidor, que é similar ao protocolo MQTT, se tem a possibilidade de realizar interoperabilidade entre os dois protocolos, fazendo com que mensagens do protocolo MQTT sejam enviadas via Websocket e vice-versa. Este protocolo é utilizado pela aplicação

4 RESULTADOS

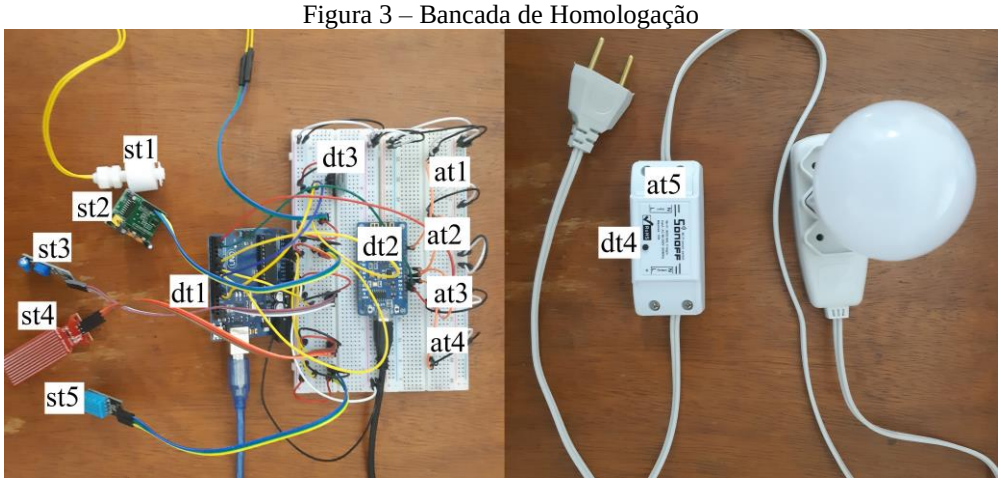
Neste capítulo são expostos os resultados do trabalho, ou seja, o aplicativo desenvolvido no capítulo 3. Como produto principal deste trabalho, apresenta-se o código fonte. O código pode ser visualizado por meio da ferramenta Bitbucket, em que encontra-se hospedado os fontes dos aplicativos relacionados ao projeto. O repositório contendo os fontes da aplicação de gestão e da aplicação de processamento, está acessível em Greuel B (2018). Nas próximas três seções, serão apresentados o conjunto de dispositivos de automação utilizados, a aplicação de interface de usuário da aplicação por meio de descrições e capturas de tela e por fim, casos de uso hipotéticos, em que uma aplicação de automação de fluxos e mensagens pode vir a ser utilizado.

4.1 DISPOSITIVOS DE AUTOMAÇÃO UTILIZADOS

Como já indicado pela seção 3.3, se adquiriu alguns dispositivos para realizar a homologação da aplicação. A Figura 3, apresenta o resultado final da utilização destes dispositivos em uma bancada de homologação. Ao longo desta seção, será explicado a montagem da bancada separados em seus principais componentes.

A Figura 3 apresenta a bancada de homologação, contendo cinco sensores, cinco atuadores e quatro dispositivos IoT. Os sensores correspondentes são identificados por: *st1*, sensor boia, coleta dados identificando se o nível da água atingiu o nível da boia; *st2*, *Passive Infrared Sensor* (PIR), coleta dados assinalando a existência de movimentação no ambiente; *st3*, sensor de vibração, coleta dados indicando a vibração no ambiente; *st4*, coleta dados indicando a umidade relativa do solo; *st5*, coleta dados indicando a umidade e temperatura relativas do ar. Os atuadores correspondentes são identificados por: *at1*, *at2*, *at3* e *at4*, interruptores, atuam permitindo a passagem ou não de eletricidade; *at5*, interruptor, atua permitindo a passagem ou não de eletricidade. Os dispositivos IoT são identificados por: *dt1*, Arduino UNO, realiza a coleta dos dados fornecidos pelos sensores; *dt2*, ESP8266, via Internet, realiza o envio dos dados dos sensores e recebe ações de atuação; *dt3*, *Bluetooth* HM-10, via BLE, realiza o envio dos dados dos sensores; *dt4*, via Internet, recebe ações de atuação.

Figura 3 – Bancada de Homologação



Fonte: elaborado pelo autor.

4.1.1 Arduino UNO

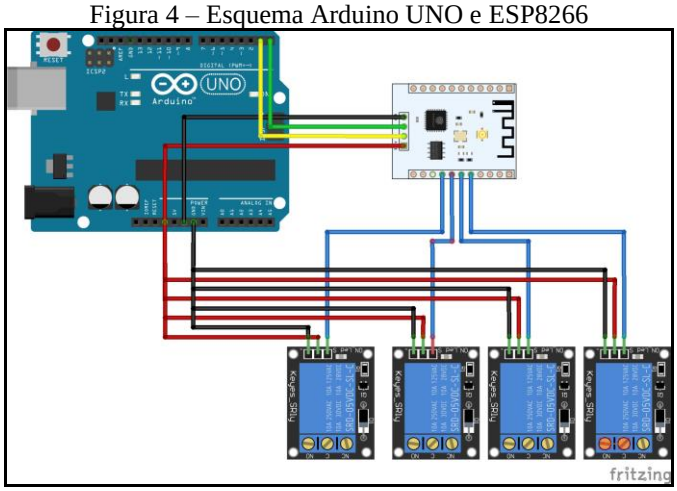
Dispositivo atua como interface de coleta de dados sensoriais, normalizando os dados recebidos dos sensores/componentes eletrônicos no protocolo de mensagem definido na seção 3.3. Após a normalização, o dispositivo envia a mensagem utilizando dois barramentos distintos, como visto no código fonte disposto no APÊNDICE B. O primeiro barramento mantém uma interface de comunicação com o dispositivo ESP8266 descrito na seção 4.1.2, neste barramento são enviados dados sensoriais unitários, intercalando entre temperatura, umidade relativa do ar, profundidade do nível de água, vibração, movimentação e por fim nível de água. O segundo barramento mantém uma interface de comunicação com o dispositivo *Bluetooth* HM-10 descrito na seção 4.1.4, neste barramento são enviados todos os dados sensoriais coletados.

Para realizar a coleta, se faz o uso de distintos componentes eletrônico. O componente DHT-11 realiza a coleta dos dados de umidade relativa do ar e de temperatura. O componente PIR realiza a coleta de movimentação no ambiente. O componente SW-420 identifica se está ocorrendo vibração no local de sua fixação.

4.1.2 ESP8266

Mantém uma interface de comunicação com o dispositivo Arduino UNO. Tem como função, realizar a ponte de comunicação entre mensagens recebidas via interface e o *broker* MQTT, também age como atuador em suas portas GPIO (entra e saída de propósito geral), ao receber mensagens de liga/desliga que, identifica a porta GPIO de atuação e seu *status*. A Figura 4 indica as conexões necessárias entre o ESP8266 e o Arduino UNO, e entre o ESP8266 e os interruptores elétricos. Após a montagem, também existe a necessidade de realizar o envio do *firmware* para o ESP8266 disponível em código fonte disposto no APÊNDICE C.

Figura 4 – Esquema Arduino UNO e ESP8266



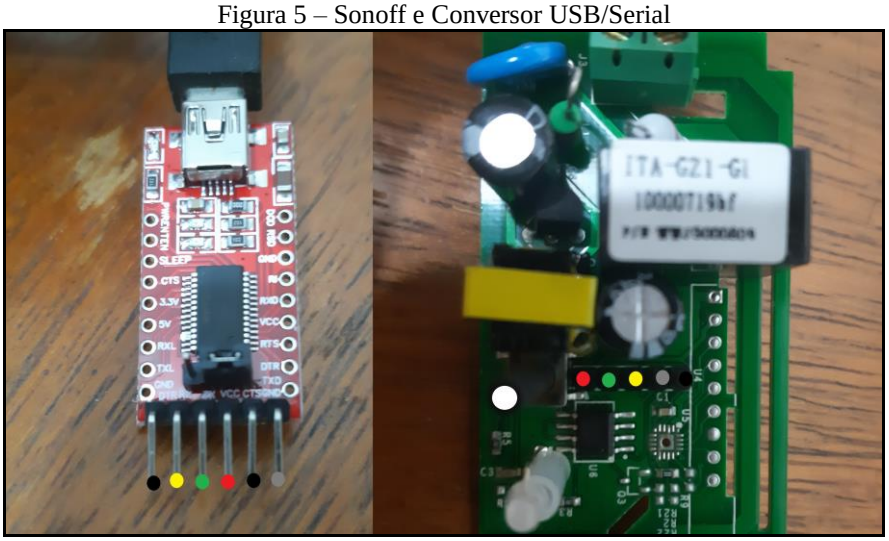
Fonte: elaborado pelo autor.

4.1.3 Sonoff

Realiza a comunicação com o *broker* MQTT, age somente como atuador em sua GPIO principal, ao receber mensagens de liga/desliga permitindo ou não a passagem de corrente elétrica. Como indicado na seção 3.3, o dispositivo Sonoff vem previamente instalado com um *firmware* proprietário, como se trata de um ESP8266, houve a troca do *firmware* para atender a aplicação neste trabalho. Para realizar a troca de *firmware* é necessário o uso de um dispositivo de conversão USB/Serial, as conexões necessárias são apresentadas na Figura 5, logo abaixo são descritos os passos necessários para realizar o envio do *firmware*.

A Figura 5 demonstra as conexões de um conversor USB/Serial a esquerda e o Sonoff a direita. Os círculos coloridos identificam a ligação entre os pinos dos dois dispositivos, os pinos identificados em cor preta, devem ser ignorados. Essas conexões permitem o envio do firmware disponível em código fonte localizado no APÊNDICE D. Antes de realizar o envio do firmware, o Sonoff necessita iniciar em modo flash, para isto, deve-se seguir os seguintes passos: desconectar o dispositivo de qualquer fonte de energia, manter pressionado o botão identificado pelo círculo branco, conectar o conversor USB/Serial na porta USB utilizada para realizar o envio do firmware, soltar o botão identificado pelo círculo branco após 10 segundos, por fim, basta enviar o *firmware*.

Figura 5 – Sonoff e Conversor USB/Serial

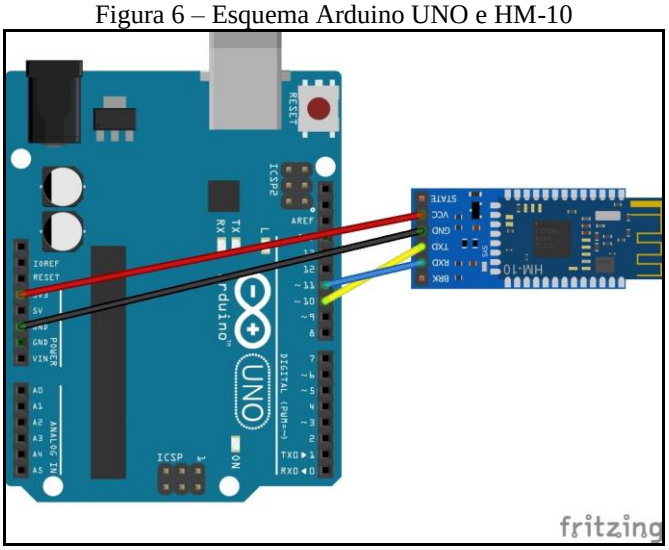


Fonte: elaborado pelo autor.

4.1.4 *Bluetooth* HM-10

Mantém uma interface de comunicação com o dispositivo Arduino UNO, realizando a ponte da mensagem recebida via interface para o dispositivo pareado. Se faz a necessidade de realizar as conexões assim como indicadas na Figura 6. A Figura 6 demonstra o esquema de conexão entre o dispositivo Arduino UNO e o dispositivo *Bluetooth* HM-10, ao contrário dos outros dispositivos, não é necessário realizar alterações no *firmware* do HM-10.

Figura 6 – Esquema Arduino UNO e HM-10

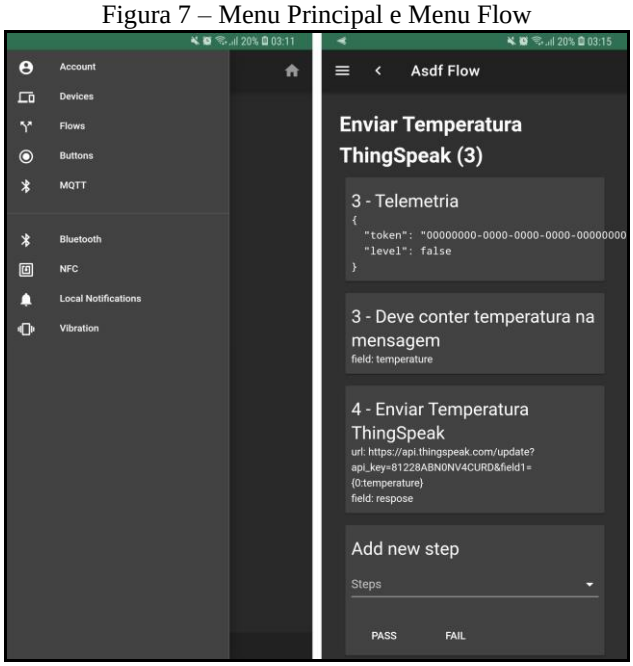


Fonte: elaborado pelo autor.

4.2 DEMONSTRAÇÃO DA APLICACAO DE INTERFACE DO USUÁRIO

Nesta seção serão apresentadas as telas da aplicação de interface do usuário, a Figura 7, apresenta a direita o menu principal da aplicação, contendo todas as páginas acessíveis, e a esquerda a funcionalidade principal da aplicação demonstrando a configuração de um fluxo. O diagrama de classes da aplicação pode ser visto no Apêndice A.

Figura 7 – Menu Principal e Menu Flow



Fonte: elaborado pelo autor.

Ao acessar o item de menu `Account`, a página da Imagem 1 (Apêndice E) será apresentada, apresentando os dados da conta do usuário, com o identificador único do usuário no campo identificado como `Id`, o nome escolhido pelo usuário no campo identificado como `Name` e o identificador único para integração com a aplicação no campo identificado como `Token`. Ao acessar o item de menu `Devices` , a página da Imagem 2 (Apêndice E) será apresentada, permite o cadastro de um novo dispositivo, utilizando o formulário identificado como `New Device`, informando o nome desejado no campo identificado como `Name`. Ao se realizar o cadastro de um novo dispositivo, a aplicação será responsável por atribuir um identificador único, chamado de *token.* O *token* deve ser utilizado como nome de registro no *broker* de mensageria. Lista os dispositivos cadastrados pelo usuário, cada dispositivo dispõe de um nome e um identificador único. Ao acessar o item de menu *Flows*, a página da Imagem 2 (Apêndice E) será apresentada,permitindo o cadastro de um novo fluxo, utilizando o formulário identificado como `New Flow`, informando o nome desejado no campo identificado como `Name`. Lista os fluxos existentes para o usuário corrente, permitindo sua edição no botão identificado como `EDIT` ou sua remoção no botão identificado como `DELETE`.

Ao acessar a edição de um `Flow`, listado na página de `Flows`, a página da direita da Figura 7 será apresentada, possibilitando a edição do fluxo selecionado, permitindo a escolha de um `gatilho`, subsequentemente permite a adição das próximas ações a serem executadas, definindo se ação deverá ser executada em falha identificado pelo botão `FAIL` ou em caso de sucesso identificado pelo botão `PASS`. A página da direita da Figura 7, apresenta um exemplo da configuração de um fluxo, onde, via recebimento de uma mensagem de protocolo MQTT, o gatilho será acionado e o `fluxo` será executado. Esta configuração apresenta três passos, sendo o primeiro, o `gatilho`, o segundo um nodo `GUARD`, garantindo a existência do atributo `temperature`, por último, um nodo `GET`, realizando o envio do atributo `temperature` para o serviço ThingSpeak.

Ao acessar o item de menu `Buttons`, a página da Imagem 3 (Apêndice E) será apresentada, listando os gatilhos descritos como `BUTTON`, permitindo o acionamento de um gatilho no botão identificado como `TRIGGER`. Ao acessar o item de menu `MQTT`, a página da Imagem 4 (Apêndice E) será apresentada, permitindo o registro do dispositivo no *broker* de mensageria, podendo assumir a identidade de qualquer outro dispositivo cadastrado. Para iniciar a comunicação com o *broker* se faz necessário acionar o botão identificado como `START`. Lista as mensagens enviadas para o dispositivo. Ao acessar o item de menu `Bluetooth`, a página da Imagem 5 (Apêndice E) será apresentada,, permitindo a conexão a um dispositivo via protocolo *bluetooth*, listando as mensagens enviadas do dispositivo pareado via protocolo *bluetooth.*

Ao acessar o item de menu NFC, a página da Imagem 6 (Apêndice E) será apresentada, permitindo a leitura e escrita de RFIDs. Lista as mensagens enviadas via broker depende do registro descrito na seção, se for desejado realizar a leitura de um RFID, deve-se acionar o botão identificado

como READ. Se for desejado realizar a escrita de um RFID, deve-se acionar o botão identificado como WRITE, enviando os dados contidos dentro de bloco. Ao acessar o item de menu `Local Notifications`, a página Imagem 7 (Apêndice E) será apresentado, permitindo que requisições de notificação sejam enviadas para o sistema operacional nativo. Ao acessar o item e menu `Vibration` a página da Imagem 8 (Apêndice E) permitindo que requisições de vibração sejam enviadas para o sistema operacional nativo.

4.3 CASOS DE USO HIPOTÉTICOS

As próximas seções apresentam três casos de usos hipotéticos, dois casos foram elaborados abordando automação agrária e um caso foi elaborado abordando automação industrial. Os três casos foram levantados e contextualizados em entrevista verbal e implementados utilizando a aplicação desenvolvida neste trabalho.

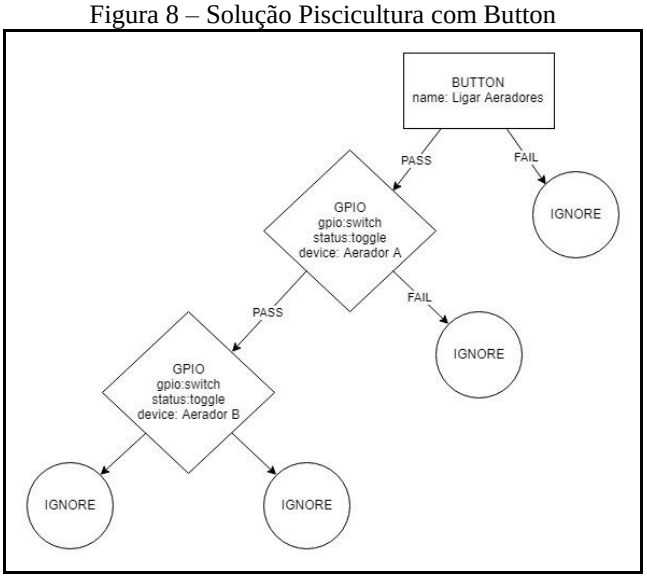
4.3.1 Automação agrária

Para Monaco (2018) o setor agrícola normalmente é mais propenso a riscos. Muitos fatores, que variam desde imprevistos de irrigação e previsões de precipitação até a má qualidade do solo e métodos de colheita/plantio, podem ter efeitos adversos sobre a produtividade geral. Porém, tal fator pode ser mitigado com a utilização da IoT, em afirmação Monaco (2018), alega que agricultores podem evitar essas situações com a ajuda da IoT. Eles podem obter dados altamente precisos e em tempo real de seus campos usando avançados agro sensores, dependendo do tipo de decisões cruciais, como (“quando colher”, “quando irrigar” etc.).

4.3.1.1 Piscicultura

Mathias (2018) piscicultor local de Jaraguá do Sul, levantou um processo realizado diariamente em função do manejo de cinco viveiros da cultura de tilápias, que estão dispostos em uma propriedade distante de sua moradia. Duas vezes ao dia, no início da manhã e no final da tarde, principalmente em períodos de extremo calor, são ligados os aeradores, responsáveis por realizar a oxigenação da água, evitando a perca da cultura por morte. Mathias (2018) questionou a possibilidade de realizar o processo de liga e desliga dos aeradores remotamente. Sobre o caso do uso descrito por Mathias (2018), se apresenta como solução hipotética, a configuração do fluxo ilustrada na Figura 8.

Figura 8 – Solução Piscicultura com Button



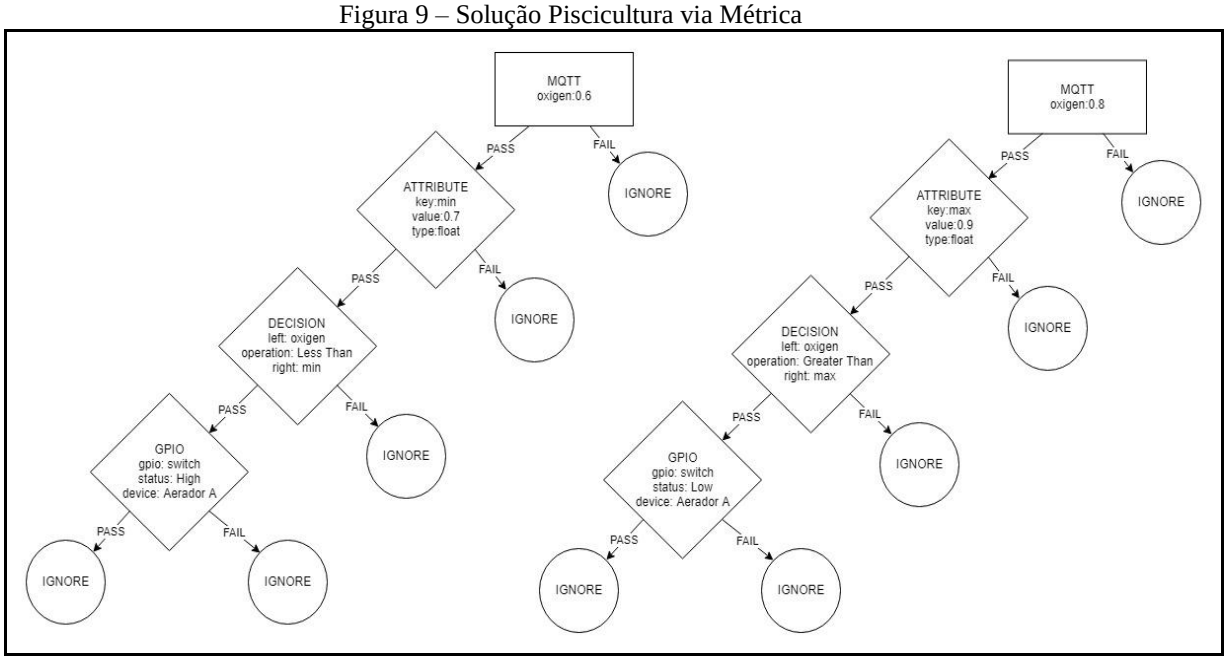
Fonte: elaborado pelo autor.

A solução apresentada na Figura 18, declara a configuração de um fluxo onde um gatilho BUTTON ativa o fluxo de envio de mensagens para alterar o *status* de uma GPIO para dois dispositivos.

A solução apresentada acima, pode ser ainda melhorada, Zaccharias e Rocha (2016, p. 59) indicam um sensor com a capacidade de realizar a coleta do oxigênio dissolvido em água, com base nos valores apresentados por Zaccharias e Rocha (2016, p. 56) a média ideal de oxigênio dissolvido para a cultura de tilápias é de 0,8mg/l, como possível configuração para esta solução, há a necessidade de dois

fluxos, um fluxo para execução da liga dos aeradores, outro fluxo para a execução da desliga dos aeradores, conforme apresentado na Figura 9.

Figura 9 – Solução Piscicultura via Métrica



Fonte: elaborado pelo autor.

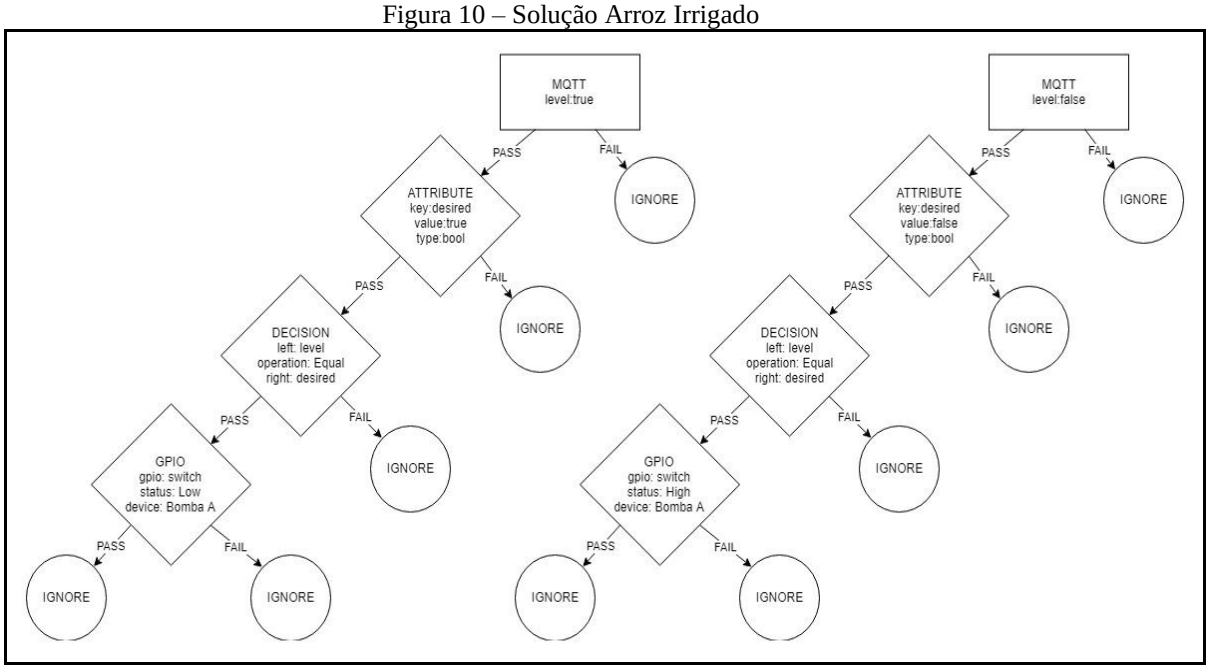
A solução apresentada na Figura 9, declara a configuração de dois fluxos. O fluxo da esquerda, responsável por realizar a liga do aerador, se declara um gatilho `MQTT` contendo a campo `oxigen`, em seguida é adicionado o campo `min` contendo o valor mínimo desejado para o nível de oxigênio, ao realizar a comparação dos dois valores no próximo passo, caso o nível de oxigênio esteja abaixo, é realizada a liga do aerador. O fluxo da direita, responsável por realizar a desliga, é muito similar, se declara um gatilho `MQTT` contendo o campo `oxigen`, adiciono um campo `max` contendo o valor máximo desejado para o nível de oxigênio, se realiza a comparação dos dois valores, caso a comparação seja verdadeira, desliga o aerador.

4.3.1.2 Cultivo de arroz irrigado

Greuel A (2018) produtor local de Jaraguá do Sul, levantou um caso de uso no cultivo de arroz irrigado, segundo Greuel A (2018) deve-se manter uma lâmina de água idealmente na altura de 8cm do caule do arroz, para isto, a solução elaborada por ele, envolve um bueiro de controle que tem como função manter a lâmina na altura desejada e a irrigação continua de água, mitigando a evaporação. Greuel A (2018) criticou a solução atual, o bueiro eventualmente entope, gerando manutenção física e a irrigação contínua gera custos financeiros em eletricidade.

Fortalecendo as afirmações de Greuel A (2018) o trabalho de Stone (2015, p. 17) avalia como uma lâmina ideal, uma altura dentre 5cm até 10cm, reforça que a evapotranspiração, deve ser contada também, no cálculo de vasão necessária de água para se manter a lâmina. No trabalho de Stone (2015, p. 37) são realizados cálculos complexos para determinar a vasão da água necessária para manter a lâmina em altura ideal, e como melhor proposta, a Figura 10 apresenta de forma simplificada o controle de nível de água, fazendo o uso de um dispositivo contendo um sensor de boia, que indica se a nível de água foi ou não atingido.

Figura 10 – Solução Arroz Irrigado



Fonte: elaborado pelo autor.

A solução é muito similar ao apresentado na seção 4.3.1.1, porém, ao invés de realizar a comparação de uma métrica, a indicação do *status* da boia, estando submersa `level:true` ou não `level:false`, já é o suficiente para realizar a liga ou desliga do sistema de irrigação.

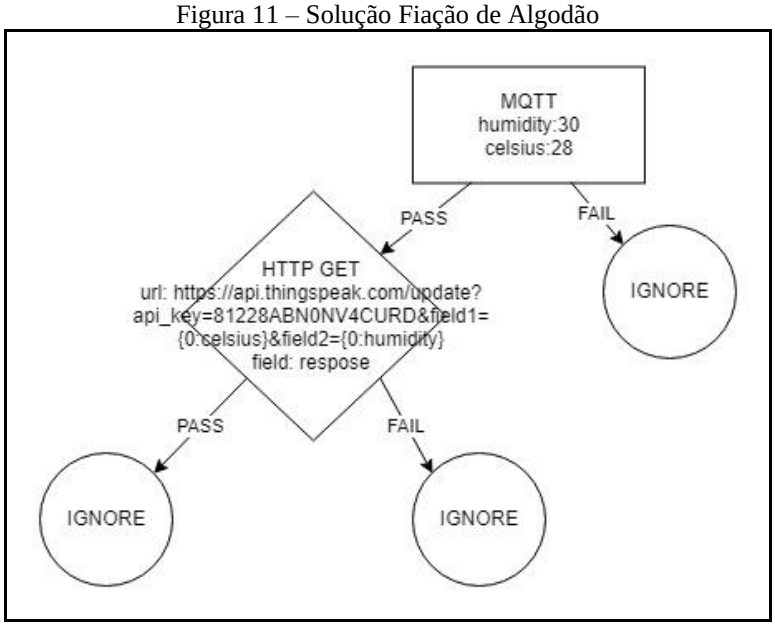
4.3.2 Automação industrial

Segundo Veroneze (2018) entre as principais vantagens oferecidas pela aplicação da Internet das coisas na indústria, se destacam o aprimoramento de diagnósticos, em que a coleta de dados torna possível detectar e analisar fatores internos e externos da produção, elaborando uma análise completa do processo identificando perdas e ganhos.

4.3.2.1 Fiação de algodão

Controle de umidade e temperatura em fiações de algodão, segundo Fagundes (2018), é o principal fator de garantia da qualidade na produção de um fio de algodão, tornando primordial o uso controlado de umidificadores e ventiladores. Em seu parque, Fagundes (2018), já dispõe de uma central de controle de umidade e calor, porém indaga que para o mesmo lote de algodão com as mesmas condições na central de controle, o fio é produzido com qualidades variadas. Para melhor identificar a causa da variação na qualidade dos fios, foi proposto realizar a coleta de umidade e temperatura, dispondo os valores em um gráfico, de forma serial. A solução para o caso, faz o uso de uma plataforma de coleta de métrica externo, a plataforma se chama ThingSpeak, o fluxo é apresentado na Figura 11.

Figura 11 – Solução Fiação de Algodão



Fonte: elaborado pelo autor.

O fluxo definido na Figura 11 demonstra o consumo de uma mensagem MQTT e o uso de dois campos dessa mensagem em uma integração externa, mostrando a possibilidade de integração com plataformas/ferramentas variadas.

5 CONCLUSÕES

Este é o capítulo de fechamento do trabalho, onde são discutidos os resultados obtidos, as decisões de implementação, as dificuldades superadas durante o desenvolvimento e projeto, e ainda são sugeridos alguns trabalhos futuros. Como principal forma de avaliação, se fez o levantamento de casos de uso, apresentando suas implementações de forma que possam ser reproduzidas, mesmo que em ambiente controlado. A funcionalidade da aplicação de fluxos e decisões, se tornou capaz de contemplar variados procedimentos em que, dado uma coleta, há a necessidade de realizar uma decisão seguida de tomada de ação, ao exemplo dos casos de uso em ambiente agrário, apresentados na seção 4.3.1. A aplicação também se mostrou apta, ao realizar integração com uma plataforma externa, como constata a seção 4.3.2. Outra meta de avaliação, foi atingida com a validação da portabilidade da aplicação de interface do usuário, a possibilidade de realizar a instalação do aplicativo de forma nativa, fazendo com que funcionalidades como *BlueTooth* e NFC fossem também acessíveis, além de possibilitar seu uso, sem a necessidade de uma conexão ativa com a Internet, reforça a viabilidade do uso de uma PWA como substituto de uma aplicação nativa. Com a apresentação da bancada de homologação, foi possível demonstrar que a solução abrange uma variedade considerável de casos de uso, além de mostrar que as definições de fluxo apresentadas neste estudo, podem ser aplicadas.

No entanto, como o *broker* de mensageria age como um centralizador e, estando somente disponível para os dispositivos conectados à rede mundial de computadores, o *broker* de mensageria se torna o principal ponto de falha. Outro caso é a falta de alguns nodos de ações que poderiam facilitar a configuração de alguns fluxos. Como extensões possíveis para este trabalho, são indicadas as seguintes opções:

a) alterar a arquitetura para que a execução do fluxo possa ocorrer em uma rede local ou internamente nos dispositivos;
b) desenvolver novos gatilhos e novos nodos de ação, de forma a atender novos casos de uso;
c) definir de maneira mais detalha o protocolo de mensagem, adicionando novos tipos de interesse ao IoT, por exemplo um tipo GPIO, definindo os estados possíveis deste tipo;
d) desenvolver um *firmware* universal compatível com o protocolo de mensagem;
e) adicionar a possibilidade de realizar o envio do *firmware* para o dispositivo utilizando a aplicação de interface de usuário e a API WebUSB.

REFERÊNCIAS

ADLINK. **A Comparison Between DDS, AMQP, MQTT, JMS, REST, CoAP, and XMPP**, [S.L], [2017]. Disponível em: <http://www.prismtech.com/sites/default/files/documents/Messaging-Whitepaper-051217.pdf>. Acesso em: 01 abr. 2018.

APPLE. **Introduction to HomeKit - Apple Developer**, [California], [2018]. Disponível em: <https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/HomeKitDeveloperGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40015050>. Acesso em: 01 abr. 2018.

ATHREYA, Arjun P.; TAGUE, Patrick. Network Self-Organization in the Internet of Things. In: IEEE International Workshop on Internet-of-Things Networking and Control, 2013, **Wireless Network and System Security Group**, 2013. p. 25-33.

BARI, Nima; MANI, Ganapathy; BERKOVICH, Simon. **Internet of Things as a Methodological Concept.** In: 2013 Fourth International Conference on Computing for Geospatial Research and Application, 2013. p. 48-55.

CERN. **Statement concerning CERN W3 software release into public domain.** [S.L], [1993]. Disponível em: <https://cds.cern.ch/record/1164399>. Acesso em: 01 abr. 2018.

CHUI, Michael; LÖFFLER, Markus; ROBERTS Roger. **The Internet of Things**. [S.L], [2010]. Disponível em: <https://www.mckinsey.com/industries/high-tech/our-insights/the-Internet-of-things>. Acesso em: 1 dez. 2018.

COETZEE, Louis; EKSTEEN, Johan; **The Internet of Things – Promise for the Future? An Introduction;** IST-Africa 2011 Conference Proceedings; 2011, p.1-9.

FAGUNDES, Ademir. **Entrevista de pesquisa sobre casos de uso**. Entrevista concedida à Ademir Fagundes. Jaraguá do Sul. 2018. Entrevista feita através de conversação – não publicada.

FLORES, Carmen Montano; LUNDMARK, Mattias; MÄHR Wolfgang. **Control vs Convenience: Critical Factors of Smart Homes**. 2005. 4f. Artigo - IT University of Göteborg, Chalmers Forskningsgången, Göteborg, 2005.

FRANSSON, Rebecca; DRIAGUINE, Alexandre. **Comparing Progressive Web Applications with Native Android Applications**. 2017. 58f. Trabalho de Conclusão de Curso - Linnaeus University, Faculty of Technology, Department of Computer Science.

GREUEL A, Sandro. **Entrevista de pesquisa sobre casos de uso**. Entrevista concedida à Sandro Greuel. Jaraguá do Sul. 2018. Entrevista feita através de conversação – não publicada.

GREUEL B, Silvio. **Asdf Flow**. Disponível em: <https://bitbucket.org/gcgfurb/silviogreuel/src/master/asdf-flow>. Acesso 5 dez. 2018.

KAPOOR, Shruti. **Progressive Web Apps 101- the What, Why and How**. [S.L], [2018]. Disponível em: <https://medium.freecodecamp.org/progressive-Web-apps-101-the-what-why-and-how-4aa5e9065ac2>. Acesso em: 27 ago. 2018.

LEPAGE, Pete. **Your First Progressive Web App**. [S.L], [2018]. Disponível em: <https://developers.google.com/Web/fundamentals/codelabs/your-first-pwapp/>. Acesso em: 27 ago. 2018.

MASTERWALKERSHOP. **Conhecendo o Sonoff Relé WiFi para Automação Residencial**, [S.l.], [2018]. Disponível em: <http://blogmasterwalkershop.com.br/automacao/conhecendo-o-sonoff-rele-wifi-para-automacao-residencial>. Acesso em: 01 dez. 2018.

MATHIAS, Paulo. **Entrevista de pesquisa sobre casos de uso**. Entrevista concedida à Paulo Mathias. Jaraguá do Sul. 2018. Entrevista feita através de conversação – não publicada.

MIT. **Bluetooth Interfacing with HM-10**, [S.l.], [2015]. Disponível em: <http://fab.cba.mit.edu/classes/863.15/doc/tutorials/programming/bluetooth.html>. Acesso em: 01 dez. 2018.

MONACO, Juliana. **4 Maneiras pelas quais IoT está revolucionando a agricultura**, [S.l.], [2017]. Disponível em: <http://www.semantix.com.br/blog/4-maneiras-pelas-quais-iot-esta-revolucionando-agricultura/>. Acesso em: 25 nov. 2018.

MOZILLA A. **HTTP**, [S.l.], [2018]. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/HTTP>. Acesso em: 24 nov. 2018.

MOZILLA B. **Service Worker API**, [S.l.], [2018]. Disponível em: <https://developer.mozilla.org/enUS/docs/Web/API/Service_Worker_API>. Acesso em: 24 nov. 2018.

MOZILLA C. **The WebSocket API**, [S.l.], [2018]. Disponível em: <https://developer.mozilla.org/enUS/docs/Web/API/WebSockets_API>. Acesso em: 24 nov. 2018.

MOZILLA D. **Using Service Workers**, [S.l.], [2018]. Disponível em: <https://developer.mozilla.org/enUS/docs/Web/API/Service_Worker_API/Using_Service_Workers>. Acesso em: 24 nov. 2018.

OLHAR DIGITAL. **A Importância do Open Source para o avanço tecnológico**, [S.L], [2016]. Disponível em: <https://olhardigital.com.br/alem_da_infra/noticia/a-importancia-do-open-source-para-o-avanco-tecnologico/61092>. Acesso em: 01 abr. 2018.

OLIVEIRA, Ricardo Rodrigues. **USO DO MICROCONTROLADOR ESP8266 PARA AUTOMAÇÃO RESIDENCIAL**. 2017. 42 f. Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia de Controle e Automação, Rio de Janeiro.

SOUZA, Marcelo Varela**. Domótica de baixo custo usando princípios de IoT**. 2016. 49f. Dissertação de Mestrado - Universidade Federal do Rio Grande do Norte, Natal, 2016.

STONE, Luís Fernando. **Eficiência do Uso da Água na Cultura do Arroz Irrigado**. Santo Antônio de Goiás - GO, 2005.

VERONEZE, Geraldo. **IoT na Indústria: por que utilizar e o que pode ser feito agora?**. [S.l.], [2018]. Disponível em: <https://www.pollux.com.br/blog/iot-na-industria-por-que-utilizar-e-o-que-pode-ser-feito>. Acesso em: 27 maio 2018.

VORAPOJPISUT, Supachai. **A Lightweight Framework of Home Automation System Based on the IFTTT Model**. 2015. 8f. Artigo - Thammasat University, Pathumthani, 2015.

W3C. **Web App Manifest**. [S.l.], [2018]. Disponível em: <https://www.w3.org/TR/appmanifest/>. Acesso em: 27 maio 2018.

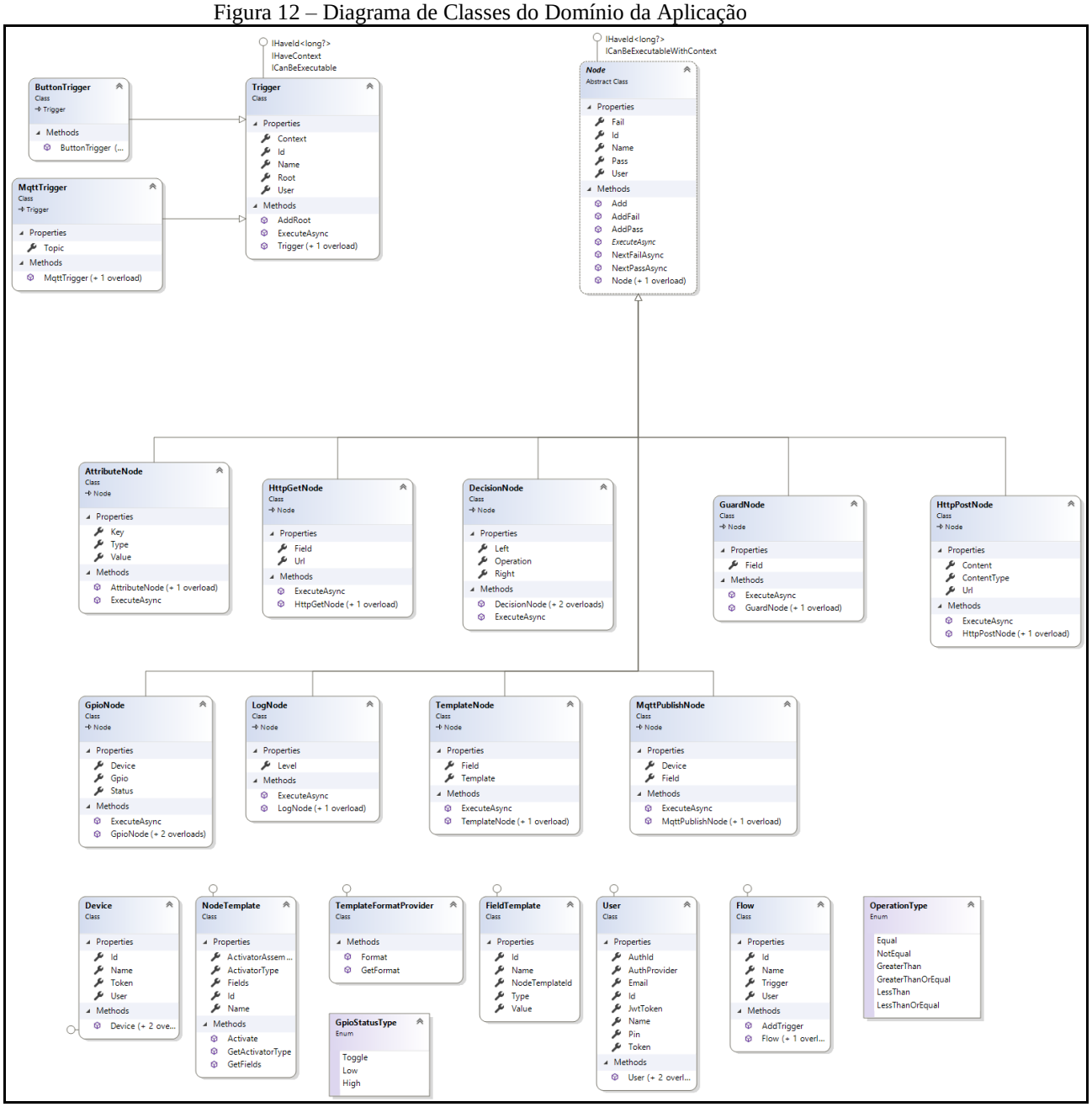
ZACCHARIAS, R. L.; DA ROCHA, R. V. Automação dos processos de produção e controle para aumento de produtividade e redução de desperdícios na piscicultura. **RECoDAF – Revista Eletrônica Competências Digitais para Agricultura Familiar**, Tupã, v. 2, n. 2, p. 52-67, jul./dez. 2016. ISSN: 2448-0452

APÊNDICE A

DIAGRAMAS DE ESPECIFICAÇÃO

Este apêndice apresenta o diagrama de classes do domínio da aplicação, identificando as classes utilizadas para o modelo de fluxo e execução (Figura 12).

Figura 12 – Diagrama de Classes do Domínio da Aplicação



Fonte: elaborado pelo autor

APÊNDICE B

CÓDIGO FONTE ARDUINO UNO

```
//#MQTT_MAX_PACKET_SIZE 255

#include <ESP8266WiFi.h>
#include <PubSubClient.h>

#define TOGGLE 3
#define DLED D7
#define DRELAY D6

//WIFI
const char* WIFI_SSID = "iot";
const char* WIFI_PASS = "12345678";

//MQTT
const char* MQTT_IP = "athletic-prawn.rmq.cloudamqp.com";
//const char* MQTT_IP = "broker.greuel.com.br";
const int   MQTT_PORT = 1883;
const char* MQTT_USER = "iot";
const char* MQTT_PASS = "iot";
const char* MQTT_DEVICE = "11111111-1111-1111-1111-111111111111";

void setup_wifi();
void reconnect_mqtt();
void setup_mqtt();
void callback_mqtt(char* topic, byte* payload, unsigned int length);
int getJustSwitch(String data);
int getPin(String data);
int getSwitch(String data);

WiFiClient client;
PubSubClient mqtt(MQTT_IP, MQTT_PORT, callback_mqtt, client);

void setup() {
        Serial.begin(115200);
  while(!Serial) { }

        //for(int pin = 0; pin <= 13; pin++) {
                pinMode(DLED, OUTPUT);
   pinMode(DRELAY, OUTPUT);
        //}

        setup_wifi();
        setup_mqtt();
}

void setup_wifi() {
```

```
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(WIFI_SSID);

  WiFi.begin(WIFI_SSID, WIFI_PASS);

  while (WiFi.status() != WL_CONNECTED) {
                digitalWrite(DLED, LOW);
    delay(500);
    Serial.print(".");
                Serial.println(WiFi.localIP());
    Serial.println(WiFi.status());
    digitalWrite(DLED, HIGH);
    delay(500);
  }
  Serial.println("");
  Serial.println("WiFi connected");

  Serial.println("Server started");

        Serial.print("ip:");
  Serial.println(WiFi.localIP());
}

void reconnect_mqtt() {
  while (!mqtt.connected()) {
    Serial.println("Connecting to MQTT...");

    if (mqtt.connect(MQTT_DEVICE, MQTT_USER, MQTT_PASS)) {
      Serial.println("mqtt connected");
    } else {
      Serial.print("failed with state ");
      Serial.print(mqtt.state());
      delay(2000);
    }
  }
}

void setup_mqtt() {
  reconnect_mqtt();
  if(mqtt.subscribe("telemetry/sensor")) {
    Serial.println("subscribed");
  }
}

void callback_mqtt(char* topic, byte* payload, unsigned int length) {
        String data = String((char*)payload);
  Serial.println(data.c_str());
        int justSwitch = getJustSwitch(data);
        int pin = getPin(data);
        int sw = getSwitch(data);

        if (justSwitch > 0) {
                digitalWriteWithToggle(DRELAY, sw);
                digitalWriteWithToggle(DLED, sw);
        } else {
                digitalWriteWithToggle(pin, sw);
        }
}
```

```
void digitalWriteWithToggle(int pin, int sw) {
        if (pin < 0 || sw < 0) return;

  if (sw == TOGGLE) {
    digitalWrite(pin, !digitalRead(pin));
  } else {
    digitalWrite(pin, sw);
  }
}

void loop() {
  reconnect_mqtt();
  mqtt.loop();
}

int getJustSwitch(String data) {
        if(data.indexOf("switch") > 1) return 1;
        return -1;
}

int getPin(String data) {
        if(data.indexOf("d0") > 1) return D0;
        if(data.indexOf("d1") > 1) return D1;
        if(data.indexOf("d2") > 1) return D2;
        if(data.indexOf("d3") > 1) return D3;
        if(data.indexOf("d4") > 1) return D4;
        if(data.indexOf("d5") > 1) return D5;
        if(data.indexOf("d6") > 1) return D6;
        if(data.indexOf("d7") > 1) return D7;
        if(data.indexOf("d8") > 1) return D8;
        if(data.indexOf("d9") > 1) return D9;
        if(data.indexOf("d10") > 1) return D10;

        return -1;
}

int getSwitch(String data) {
        if(data.indexOf("high") > 1) return HIGH;
        if(data.indexOf("low") > 1) return LOW;
  if(data.indexOf("toggle") > 1) return TOGGLE;
        return -1;
}
```

APÊNDICE C

CÓDIGO FONTE ESP8266

```
//#define MQTT_MAX_PACKET_SIZE 255

#include <ESP8266WiFi.h>
#include <PubSubClient.h>

#define D0   16
#define D1   5
#define D2   4
#define D3   0
#define D4   2
#define D5   14
#define D6   12
#define D7   13
#define D8   15
#define D9   3
#define D10  1
#define DLED D4
#define DRELAY D5

#define TOGGLE 3

//WIFI
const char* WIFI_SSID = "iot";
const char* WIFI_PASS = "12345678";

//MQTT
const char* MQTT_IP = "athletic-prawn.rmq.cloudamqp.com";
//const char* MQTT_IP = "broker.greuel.com.br";
const int   MQTT_PORT = 1883;
const char* MQTT_USER = "iot";
const char* MQTT_PASS = "iot";
const char* MQTT_DEVICE = "22222222-2222-2222-2222-222222222222";

void setup_wifi();
void reconnect_mqtt();
void setup_mqtt();
void callback_mqtt(char* topic, byte* payload, unsigned int length);
int getJustSwitch(String data);
int getPin(String pin);
int getSwitch(String sw);

WiFiClient client;
PubSubClient mqtt(MQTT_IP, MQTT_PORT, callback_mqtt, client);

void setup() {
        Serial.begin(115200);
  while(!Serial) { }

        pinMode(D0 , OUTPUT);
        pinMode(D1 , OUTPUT);
        pinMode(D2 , OUTPUT);
        pinMode(D3 , OUTPUT);
        pinMode(D4 , OUTPUT);
        pinMode(D5 , OUTPUT);
        pinMode(D6 , OUTPUT);
        pinMode(D7 , OUTPUT);
```

```
        pinMode(D8 , OUTPUT);
        //pinMode(D9 , OUTPUT);
        //pinMode(D10, OUTPUT);

        setup_wifi();
        setup_mqtt();
}

void setup_wifi() {
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(WIFI_SSID);

  WiFi.begin(WIFI_SSID, WIFI_PASS);

  while (WiFi.status() != WL_CONNECTED) {
                digitalWrite(DLED, LOW);
    delay(500);
    Serial.print(".");
                Serial.println(WiFi.localIP());
    Serial.println(WiFi.status());
    digitalWrite(DLED, HIGH);
    delay(500);
  }
  Serial.println("");
  Serial.println("WiFi connected");

  Serial.println("Server started");

        Serial.print("ip:");
  Serial.println(WiFi.localIP());
}

void reconnect_mqtt() {
  while (!mqtt.connected()) {
    Serial.println("Connecting to MQTT...");

    if (mqtt.connect(MQTT_DEVICE, MQTT_USER, MQTT_PASS)) {
      Serial.println("mqtt connected");
    } else {
      Serial.print("failed with state ");
      Serial.print(mqtt.state());
      delay(2000);
    }
  }
}

void setup_mqtt() {
  reconnect_mqtt();
  if(mqtt.subscribe("#")) {
    Serial.println("subscribed");
  }
}

void callback_mqtt(char* topic, byte* payload, unsigned int length) {
        String data = String((char*)payload);
  Serial.println(data.c_str());
  int justSwitch = getJustSwitch(data);
  int pin = getPin(data);
```

```
            int sw = getSwitch(data);

  if (justSwitch > 0) {
    digitalWriteWithToggle(DRELAY, sw);
    digitalWriteWithToggle(DLED, sw);
  } else {
    digitalWriteWithToggle(pin, sw);
  }
}

void digitalWriteWithToggle(int pin, int sw) {
  if (pin < 0 || sw < 0) return;

  if (sw == TOGGLE) {
    digitalWrite(pin, !digitalRead(pin));
  } else {
    digitalWrite(pin, sw);
  }
}

void loop() {
  reconnect_mqtt();
  mqtt.loop();

  if(Serial.available()) {
    String payload = Serial.readStringUntil('!');
    if(!mqtt.publish("telemetry", payload.c_str(), true)) {
      Serial.println("Telemetry not sended");
    }
    Serial.println(payload.c_str());
  }
}

int getJustSwitch(String data) {
  if(data.indexOf("switch") > 1) return 1;
  return -1;
}

int getPin(String pin) {
            if(pin.indexOf("d0") > 1) return D0;
            if(pin.indexOf("d1") > 1) return D1;
            if(pin.indexOf("d2") > 1) return D2;
            if(pin.indexOf("d3") > 1) return D3;
            if(pin.indexOf("d4") > 1) return D4;
            if(pin.indexOf("d5") > 1) return D5;
            if(pin.indexOf("d6") > 1) return D6;
            if(pin.indexOf("d7") > 1) return D7;
            if(pin.indexOf("d8") > 1) return D8;
  if(pin.indexOf("d9") > 1) return D9;
  if(pin.indexOf("d10") > 1) return D10;

            return -1;
}

int getSwitch(String sw) {
            if(sw.indexOf("high") > 1) return HIGH;
            if(sw.indexOf("low") > 1) return LOW;
  if(sw.indexOf("toggle") > 1) return TOGGLE;
            return -1;
}
```

APÊNDICE D

CÓDIGO FONTE SONOFF

```
//#MQTT_MAX_PACKET_SIZE 255

#include <ESP8266WiFi.h>
#include <PubSubClient.h>

#define TOGGLE 3
#define DLED D7
#define DRELAY D6

//WIFI
const char* WIFI_SSID = "iot";
const char* WIFI_PASS = "12345678";

//MQTT
const char* MQTT_IP = "athletic-prawn.rmq.cloudamqp.com";
//const char* MQTT_IP = "broker.greuel.com.br";
const int   MQTT_PORT = 1883;
const char* MQTT_USER = "iot";
const char* MQTT_PASS = "iot";
const char* MQTT_DEVICE = "11111111-1111-1111-1111-111111111111";

void setup_wifi();
void reconnect_mqtt();
void setup_mqtt();
void callback_mqtt(char* topic, byte* payload, unsigned int length);
int getJustSwitch(String data);
int getPin(String data);
int getSwitch(String data);

WiFiClient client;
PubSubClient mqtt(MQTT_IP, MQTT_PORT, callback_mqtt, client);

void setup() {
        Serial.begin(115200);
  while(!Serial) { }

        //for(int pin = 0; pin <= 13; pin++) {
                pinMode(DLED, OUTPUT);
    pinMode(DRELAY, OUTPUT);
        //}

        setup_wifi();
        setup_mqtt();
}

void setup_wifi() {
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(WIFI_SSID);

  WiFi.begin(WIFI_SSID, WIFI_PASS);

  while (WiFi.status() != WL_CONNECTED) {
                digitalWrite(DLED, LOW);
```

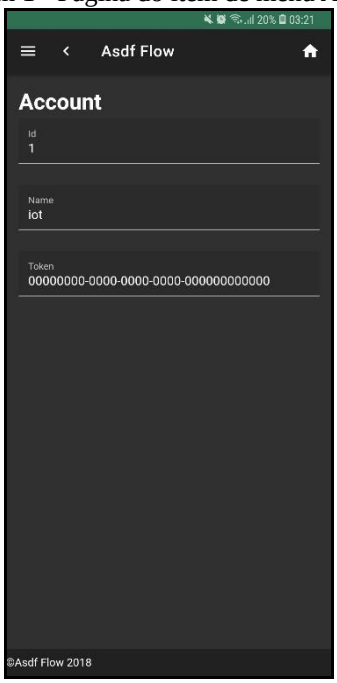
```
    delay(500);
    Serial.print(".");
                        Serial.println(WiFi.localIP());
    Serial.println(WiFi.status());
    digitalWrite(DLED, HIGH);
    delay(500);
  }
  Serial.println("");
  Serial.println("WiFi connected");

  Serial.println("Server started");

            Serial.print("ip:");
  Serial.println(WiFi.localIP());
}

void reconnect_mqtt() {
  while (!mqtt.connected()) {
    Serial.println("Connecting to MQTT...");

    if (mqtt.connect(MQTT_DEVICE, MQTT_USER, MQTT_PASS)) {
      Serial.println("mqtt connected");
    } else {
      Serial.print("failed with state ");
      Serial.print(mqtt.state());
      delay(2000);
    }
  }
}

void setup_mqtt() {
  reconnect_mqtt();
  if(mqtt.subscribe("telemetry/sensor")) {
    Serial.println("subscribed");
  }
}

void callback_mqtt(char* topic, byte* payload, unsigned int length) {
            String data = String((char*)payload);
  Serial.println(data.c_str());
            int justSwitch = getJustSwitch(data);
            int pin = getPin(data);
            int sw = getSwitch(data);

            if (justSwitch > 0) {
                        digitalWriteWithToggle(DRELAY, sw);
                        digitalWriteWithToggle(DLED, sw);
            } else {
                        digitalWriteWithToggle(pin, sw);
            }
}

void digitalWriteWithToggle(int pin, int sw) {
            if (pin < 0 || sw < 0) return;

  if (sw == TOGGLE) {
    digitalWrite(pin, !digitalRead(pin));
  } else {
    digitalWrite(pin, sw);
  }
```

```
}

void loop() {
  reconnect_mqtt();
  mqtt.loop();
}

int getJustSwitch(String data) {
        if(data.indexOf("switch") > 1) return 1;
        return -1;
}

int getPin(String data) {
        if(data.indexOf("d0") > 1) return D0;
        if(data.indexOf("d1") > 1) return D1;
        if(data.indexOf("d2") > 1) return D2;
        if(data.indexOf("d3") > 1) return D3;
        if(data.indexOf("d4") > 1) return D4;
        if(data.indexOf("d5") > 1) return D5;
        if(data.indexOf("d6") > 1) return D6;
        if(data.indexOf("d7") > 1) return D7;
        if(data.indexOf("d8") > 1) return D8;
        if(data.indexOf("d9") > 1) return D9;
        if(data.indexOf("d10") > 1) return D10;

        return -1;
}

int getSwitch(String data) {
        if(data.indexOf("high") > 1) return HIGH;
        if(data.indexOf("low") > 1) return LOW;
  if(data.indexOf("toggle") > 1) return TOGGLE;
        return -1;
}
```

APÊNDICE E

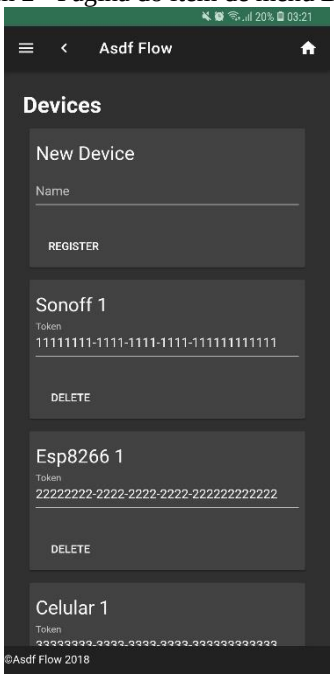
IMAGENS DAS TELAS DA APLICAÇÃO DE INTERFACE DO USUÁRIO

Imagem 1 - Página do item de menu *Account*



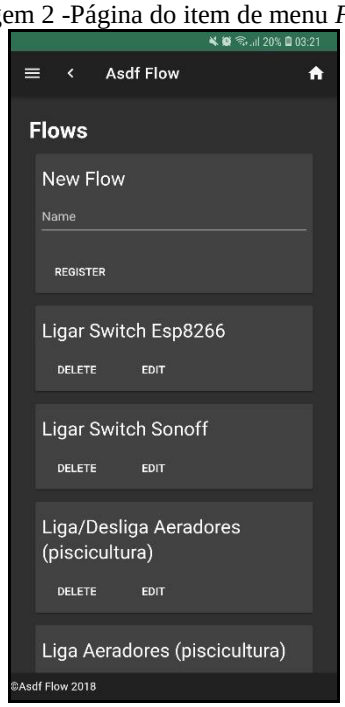
Fonte: Elaborado pelo autor.

Imagem 2 - Página do item de menu *Devices*



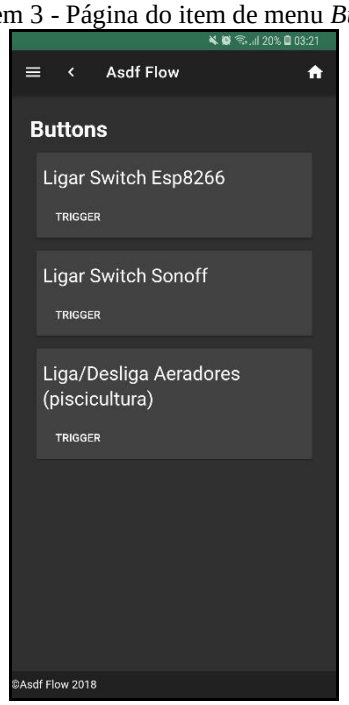
Fonte: Elaborado pelo autor.

Imagem 2 -Página do item de menu *Flows*



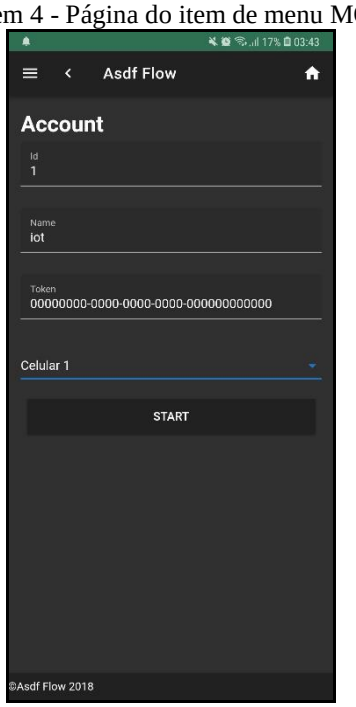
Fonte: Elaborado pelo autor.

Imagem 3 - Página do item de menu *Buttons*



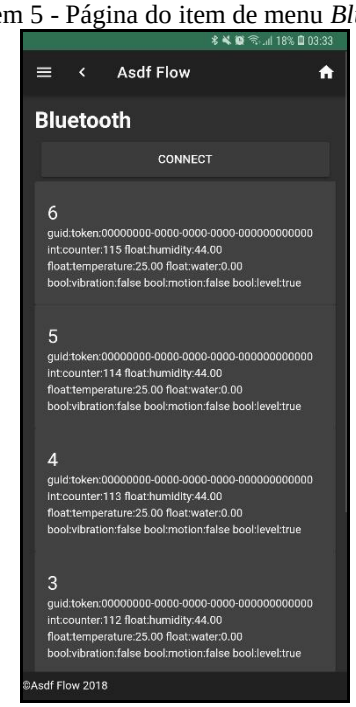
Fonte: Elaborado pelo autor.

Imagem 4 - Página do item de menu MQTT



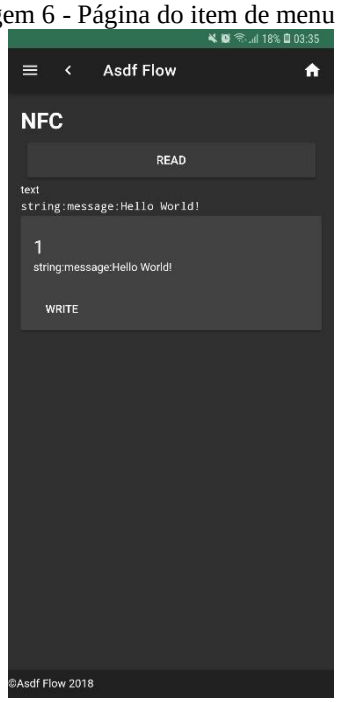
Fonte: Elaborado pelo autor.

Imagem 5 - Página do item de menu *Bluetooth*



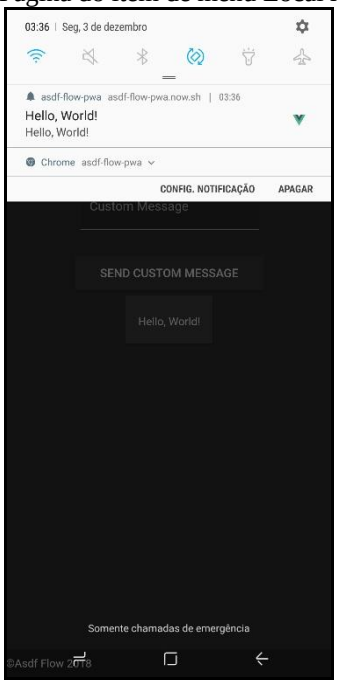
Fonte: Elaborado pelo autor.

Imagem 6 - Página do item de menu NFC



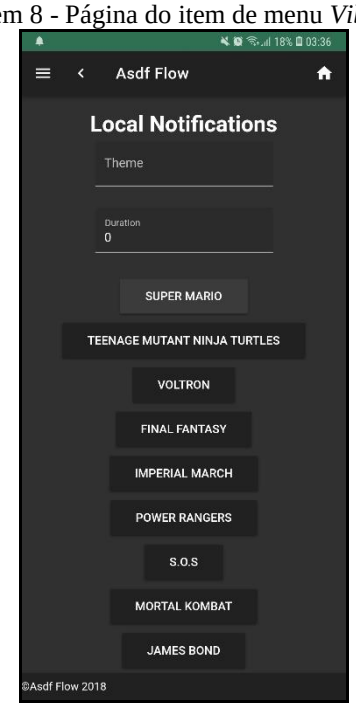
Fonte: Elaborado pelo autor.

Imagem 7 - Página do item de menu Local *Notifications*



Fonte: Elaborado pelo autor.

Imagem 8 - Página do item de menu *Vibration*



Fonte: Elaborado pelo autor.