UNIVERSIDADE DO ESTADO DO AMAZONAS
ESCOLA SUPERIOR DE TECNOLOGIA
GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

RAPHAEL DE SOUZA NUNES

PROJETO DE UM SISTEMA DE ABASTECIMENTO PLUVIAL BASEADO EM *IOT*

MANAUS/AM
2021

RAPHAEL DE SOUZA NUNES

PROJETO DE UM SISTEMA DE ABASTECIMENTO PLUVIAL BASEADO EM *IOT*

Trabalho de Conclusão de Curso apresentado à banca avaliadora do curso de Engenharia de Controle e Automação da Universidade do Estado do Amazonas - UEA, unidade Escola Superior de Tecnologia - EST, como pré-requisito para obtenção do título de Bacharel em Engenharia de Controle e Automação.

Orientador: **Prof. Dr. Israel Mazaira Morales**
Universidade do Estado do Amazonas

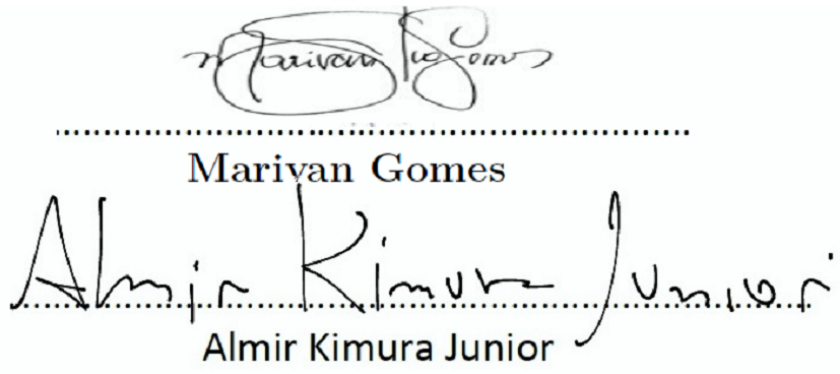
MANAUS/AM
2021

RAPHAEL DE SOUZA NUNES

PROJETO DE UM SISTEMA DE ABASTECIMENTO PLUVIAL BASEADO EM *IOT*

Trabalho de Conclusão de Curso apresentado à banca avaliadora do curso de Engenharia de Controle e Automação da Universidade do Estado do Amazonas - UEA, unidade Escola Superior de Tecnologia - EST, como pré-requisito para obtenção do título de Bacharel em Engenharia de Controle e Automação.

Aprovado por:



Prof. Dr. Daniel Guzmán del Río
Membro - EST/UEA

MANAUS/AM
2021

AGRADECIMENTOS

Agradeço a Deus pelo dom da vida e do conhecimento os quais me proporcionaram chegar até aqui.

Agradeço à minha família por todo apoio, paciência e dedicação contribuindo diretamente para que eu pudesse traçar o meu caminho de forma mais fácil durante todos esses anos.

Agradeço aos meus pais, Wagner Antônio da Silva Nunes e Mariluce de Souza Nunes, que sempre me incentivam e me guiaram pelos meus caminhos do conhecimento.

Agradeço todos os meus professores da Universidade do Estado do Amazonas, em especial, os professores: Israel Mazaira, Moisés Bastos, Charles Melo, Jozias Parente, Raimundo Cláudio, Israel Torné e Fábio Cardoso os quais não mediram esforços para transmitir o conhecimento sejam eles em sala de aula, em projetos ou mesmo em conversas pelos corredores da universidade.

Agradeço a todos os meus amigos da universidade, em especial, Michael Sidi, Mateus Ramos, Samuel Marinho, Rubens Andrade e Isaque Vilson que sempre estiveram ao meu lado, pela amizade incondicional e pelo apoio demonstrado.

Por último, quero agradecer também à Universidade do Estado do Amazonas e todo o seu corpo docente.

RESUMO

Com base na busca da produção de tecnologias para diminuir a degradação do meio ambiente e, unindo-se a isto, a tendência pela implementação de *Smart Things*, dentro do âmbito do *IoT - Internet Of Things*, o projeto desenvolvido visou a elaboração de um sistema para criação de uma Cisterna Automatizada a qual conta com um protótipo físico para simulação da coleta e distribuição de água da chuva, integrando as três áreas de um projeto *IoT*: *Hardware*, *Firmware* e *Software*. No segmento de *Firmware*, por meio do uso de microcontroladores e microprocessadores em integração com sensores, atuadores e a *Intranet*, será possível coletar dados e realizar ações em tempo real oriundas de comandos do usuário, assim como receber atualizações para melhoria contínua do *Software* Embarcado (*Firmware Update*). Na seção de *Software*, a utilização de tecnologias para modelagem de aplicações *Mobile* e *Desktop*, como *React Native* e *Electron*, garantiram a criação de interfaces amigáveis para rotinas de cadastro, atualização, monitoramento e controle dos processos do sistema de coleta. Por fim, no campo de *Hardware*, elaborou-se esquemáticos possibilitando a criação de *layout's*, que posteriormente, garantirão a fabricação de placas de circuito impresso - *PCB's*. Com intuito de validação, o protótipo construído possui estruturas de: alimentação, conversão de sinais, comunicação, controle e acionamento.

Palavras-chave: *IoT - Internet of Things. Cisterna automatizada. Smart Things. Intranet. Firmware. Hardware. Software. React Native. Electron.*

ABSTRACT

Based on the search for the production of technologies to reduce the degradation of the environment and, joining this, the trend towards the implementation of Smart Things, within the scope of IoT - Internet Of Things, the developed project aimed at the elaboration of a system to create an Automated Cistern which has a physical prototype for simulating the collection and distribution of rainwater, integrating the three areas of an IoT project: Hardware, Firmware and Software. No Firmware segment, through the use of microcontrollers and microprocessors in integration with sensors, actuators and the Intranet, it will be possible to collect data and perform real-time actions from user commands, as well as updates for continuous improvement of the Embedded Software (Firmware To update). In the Software section, the use of technologies for modeling Mobile and Desktop applications, such as React Native and Electron, ensured the creation of user-friendly interfaces for routines for registration, updating, monitoring and control of collection system processes. Finally, in the Hardware section, schematics were created, enabling the creation of layouts, which will later guarantee the manufacture of printed circuit boards - PCB's. With the purpose of validation, the built prototype has structures for: power, signal conversion, communication, control and activation.

Keywords: *IoT - Internet of Things. Smart Things. Intranet. Firmware. Hardware. Software. React Native. Electron.*

LISTA DE FIGURAS

LISTA DE SIGLAS

IoT	Internet Of Things
PCB	Printed Circuit Board
SMCC	Sistema de Medição e Controle da Cisterna
SMCCD	Sistema de Medição e Controle da Caixa D'água
MQTT	Message Queuing Telemetry Transport
HTTP	Hypertext Transfer Protocol
GPIO	General Purpose Input Output
HAS	Home Automation System
DC	Direct Current
AC	Alternate Current
PWM	Pulse-Width Modulation
NF	Normalmente Fechada
IDE	Integrated Development Environment
CLI	Command Line Interface
GUI	Graphic User Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheet

SUMÁRIO

1 INTRODUÇÃO

A água é um recurso básico para a sustentação humana em um ambiente e ao longo do tempo várias civilizações evoluíram e padeceram em função de sua relação de uso com este recurso. Nos últimos anos, crises relacionadas ao abastecimento e à qualidade da água potável têm sido observadas em todo o Globo. Tomando o Brasil como exemplo, percebe-se uma distribuição desigual: No norte do pais há grandes reservas de água, porém nas regiões Nordeste e Sudeste há problemas de escassez e poluição dos rios.

A região norte, possuindo a maior reserva de água potável do Brasil e também os maiores índices de precipitação, é a região que possui as taxas mais altas de desperdício (GLOBO.COM, 2021). O desperdício pode ser encontrado no ambiente doméstico e nas várias etapas de: coleta, armazenamento, processamento e principalmente na distribuição do recurso.

A instalação de uma cisterna garante, pelo menos, três pontos positivos: que seja possível utilizar a água de precipitações para afazeres domésticos, reduzindo o consumo mensal de determinada residência; pode diminuir o desperdício durante a etapa de distribuição da concessionária e contribuir para a redução da incidência de inundações em grandes cidades, uma vez que grande parte dessa água não seria descartada, mas armazenada.

A aplicação de uma cisterna automatizada garante uma supervisão do nível de água constantemente assim como o controle/acionamento de bombas para alimentação de tanques ou caixas d'agua para tarefas específicas trazendo comodidade e fomentando os motivos de aplicação.

2 PROBLEMÁTICA

Segundo a Gartner, Inc BizMeet (BIZMEET, 2017) desde 2017 existem mais objetos conectados à Internet do que as 7 bilhões de pessoas no mundo. Isso demonstra uma crescente busca pela obtenção e controle das mais diversas tarefas. Os objetos e dispositivos estão adquirindo funcionalidades as quais se identificam com os ramos da *IoT*, tornando-se possível a concepção de diversas melhorias no funcionamento, através da obtenção e distribuição de dados. A utilização da **Internet das Coisas** está tão conectada ao nosso cotidiano que ocasiona o nascimento de tecnologias necessariamente conectadas, como *Smart Home* e *Smart Buildings*.

Diante disso, em ambientes como na região Amazônica, onde há altos índices de precipitação, a implantação de cisternas para utilização da água da chuva se torna bastante viável. Sabe-se que a água da chuva não é própria para o consumo e para o preparo de alimentos, porém a mesma pode ser utilizada em afazeres como limpeza de locais e também em descargas de banheiros, onde há o maior nível de desperdício.

Em virtude do que foi mencionado, a aplicação de uma cisterna se torna algo viável para a economia de água ajudando na conservação do meio ambiente, porém, é possível elaborar um projeto base para ser aplicado a diversas situações? Quais ferramentas são necessárias para realizar as medições de nível? Como fazer um sistema o qual os dados poderão ser acessados remotamente? Quais cuidados se deve ter ao implantar tal sistema?

3 OBJETIVOS

3.1 Objetivo geral

O objetivo desse trabalho de conclusão de curso é a elaboração de um projeto base para aplicação de um sistema automatizado de coleta, armazenamento e distribuição de água da chuva com base em uma cisterna pluvial. Realizando medições de volume, através de sensores, acionamento de válvulas solenoides e motobombas por meio de comandos via interfaces *desktop* e *mobile*. A elaboração também contará com a criação de circuitos esquemáticos para desenvolvimento de placas de circuito impresso - *PCB's*, o uso de microcontoladores e microprocessadores em conjunto com diretrizes de Internet das Coisas - *IoT*, assim como o uso de *Frameworks* atuais baseados em *Javascript* para criação de Interfaces Homem Máquina.

3.2 Objetivos específicos

(a) Elaborar um módulo denominado **CCM - *Cistern Control Module*** para aplicação no reservatório principal, realizando medições de nível, acionamento de *motobomba*, direcionamento do fluxo de água em conexão com outros dispositivos via tecnologia *Wi-Fi*;

(b) Elaborar um módulo denominado **TCM - *Tank Control Module*** para aplicação no reservatório auxiliar, realizando medições de nível, direcionamento do fluxo de água em conexão com outros dispositivos via tecnologia *Wi-Fi*;

(c) Elaborar um aplicação *Android Mobile* (denominada **RCS APP**) e *Desktop* (denominada **RCS DESKTOP**) para executar as ações: ativação e desativação de uma bomba d'água, direcionamento do fluxo de água, visualização de dados provenientes de sensores e estruturando a possibilidade de definir as condições que executem rotinas de acionamento automático;

(d) Configurar um sistema operacional embarcado conciso (baseado em *kernel Linux*) aplicando-o a um microprocessador. O dispositivo deve possuir conexão com a *Intranet* e servir como uma central de controle e armazenamento de dados, bem como ser hospedeiro do serviço de *Broker MQTT*;

(e) Incluir no sistema, rotinas de leitura de sensores e acionamento de motobombas;

(f) Organizar um repositório *online* para realização de futuras atualizações (*upgrades*) do *firmware* embarcado.

4 HIPÓTESE E JUSTIFICATIVA

4.1 Hipótese

A inclusão da *IoT* nos mais diversos processos nos traz múltiplos benefícios, como: obter maiores informações e atuar sobre tais processos. Automatizar tarefas que são, muitas vezes repetitivas ou mesmo insalubres faz com que as tecnologias denominadas *Smart* se tornem cada vez mais um foco de pesquisa e desenvolvimento.

No âmbito do meio ambiente, a implantação de uma cisterna garante um menor desperdício de água, utilizando-a para tarefas específicas assim como uma economia no consumo. Porém a implantação da mesma, não é atraente para todos, pois existe a necessidade do trabalho braçal assim como a constante análise da quantidade de água disponível.

Esse trabalho de conclusão de curso busca elaborar um projeto base para a aplicação de uma cisterna automatizada, situando as tecnologias, os dispositivos e as ferramentas necessárias para executar o controle por meio de *smartphones* e computadores. Tais tecnologias poderão tornar a ideia de aplicação mais atraente para possíveis investimentos, em caso do projeto se tornar base de um produto.

4.2 Justificativa

Buscando estar de acordo com a convergência de produtos e processos conectados em rede, percebeu-se a necessidade e oportunidade da utilização de conceitos e tecnologias emergentes nos contextos atuais, como *Smart Things*. A aplicação de serviços que unam a busca pela preservação ambiental e conceitos citados anteriormente se destacam por possibilitar uma gama de vantagens em qualquer processo.

O presente trabalho se justifica pelos altos índices de precipitação na região do Estado do Amazonas, onde será aplicado. A coleta de água da chuva acarretará, além da diminuição do consumo de água (trazendo economia para o usuário e ajudando na preservação ambiental), em uma distribuição mais inteligente da água não potável: que pode ser utilizada para limpezas, descargas e até irrigações; e em uma motivação para investimentos futuros reduzindo, ainda mais, os gastos de implementação.

O projeto trará motivações e investidores para aplicação de novas pesquisas focadas em contextos diferentes dentro do estado, onde há: **I**. lugares que não possuem estruturas de rede ou que demandam comunicação à grandes distâncias: utilização de tecnologias como *LoRaWan*, *SigFox* e *Zigbee*; **II.** lugares que não possuem energia elétrica: união dos conceitos desse trabalho com fontes renováveis, como alimentação fotovoltaica.

5 TRABALHOS RELACIONADOS

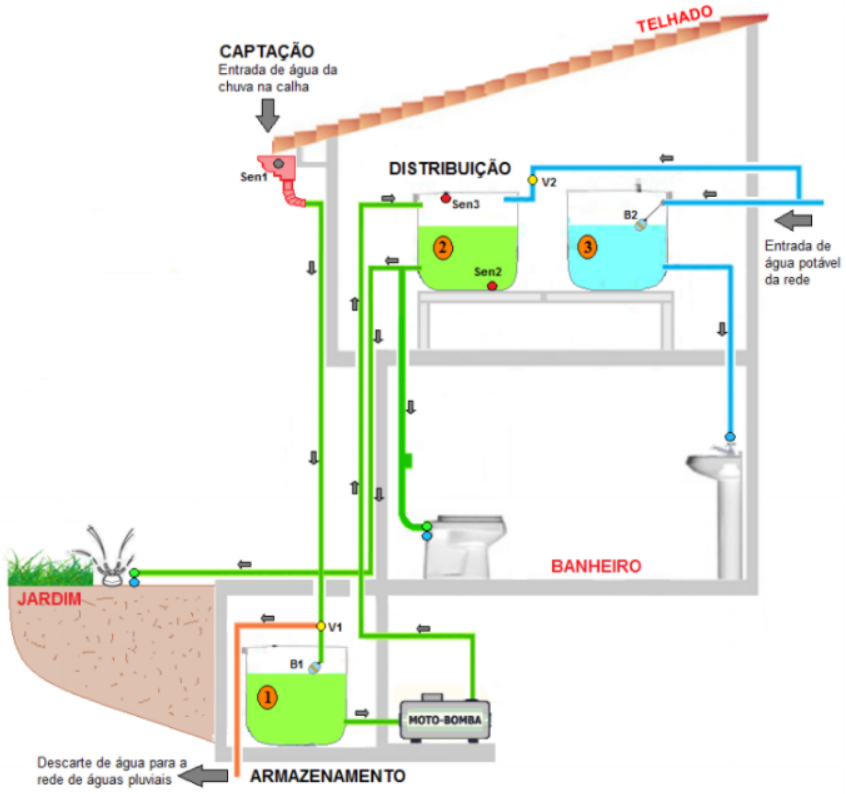
5.1 Automação e Controle em Sistema de Aproveitamento de Água da Chuva para Fins Não Potáveis

Autora: Lilia Rodrigues Lucas Magalhães

Este trabalho de conclusão de curso realizado por Lilia Rodrigues Lucas Magalhães pelo Centro Universitário de Brasília - UniCEUB apresenta uma protosta muito semelhante ao trabalho a ser desenvolvido.

A ideia deste projeto foi demonstrar o funcionamento automatizado de um sistema (representado na Figura 5.1) de aproveitamento de água de chuva por meio da elaboração de um protótipo composto pela maquete de um banheiro, confeccionada em madeira, juntamente com três reservatórios em acrílico. Esse cenário representa o sistema de distribuição de água de chuva dentro de uma residência e destina-se à utilização em descarga sanitária.

Figura 5.1 – Corte Esquemático do Sistema de Aproveitamento Proposto.



O projeto, implementado fisicamente e virtualmente no *software Proteus* (Figura 5.2), foi baseado para uma aplicação sem a conexão com redes remotas e utilizando componentes eletrônicos mais simples, como portas analógicas. O microcontrolador utilizado foi o *Philips 80C552* acoplado ao Kit *CW 552*, considerado defasado atualmente. Esse trabalho fomentou a ideia da utilização de válvulas solenoides para direcionamento do fluxo de água, assim como métodos para medição de nível e uma abordagem sobre as rotinas de funcionamento do microcontrolador.

Figura 5.2 – Simulação elaborada no *Software Proteus*.

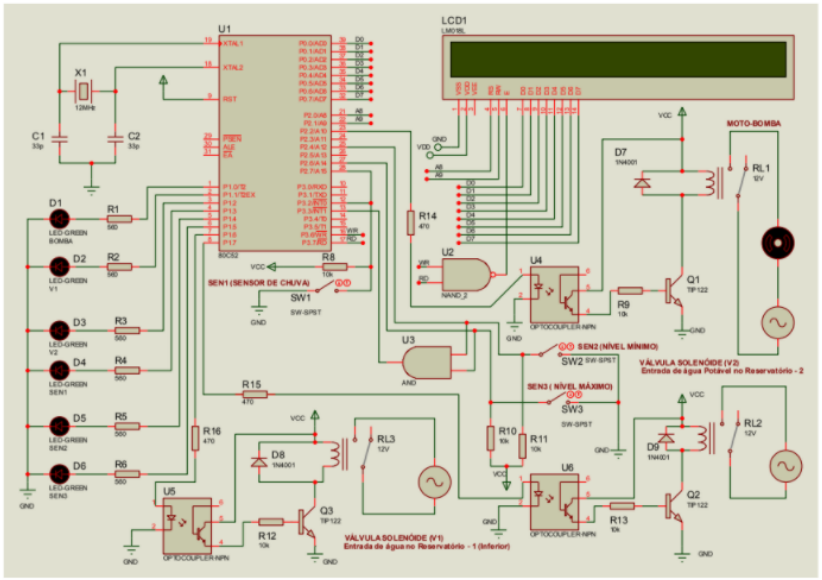
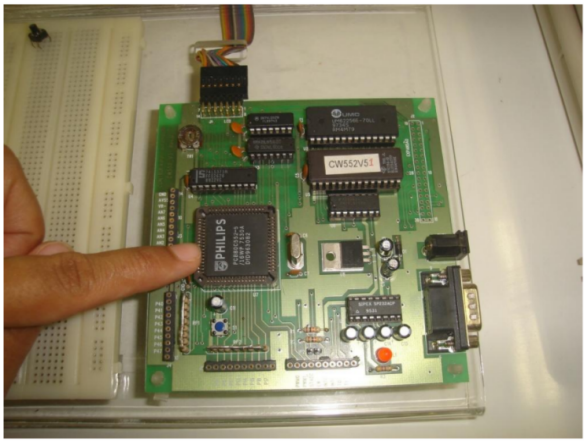


Figura 5.3 – Microcontrolador *Philips 80C552* acoplado ao kit *CW 552*.



5.2 Automação residencial Usando protocolo *MQTT*, *NODE-RED* e *Mosquitto Broker* com *ESP32* e *ESP8266*

Autor: Victor Ferreira Martins

Este trabalho de conclusão de curso desenvolvido por Victor Ferreira Martins teve o objetivo de propor uma solução de automação residencial que fosse capaz de integrar três diferentes sistemas em uma casa: monitoramento de temperatura, alarmes de segurança e acionamento de iluminação e tomadas. O objetivo dessa integração foi ajudar a alcançar três dos grandes objetivos da automação residencial: conforto, segurança e economia de energia. Como caraterísticas importantes destacam-se: a utilização do protocolo *MQTT* para comunicação através do *Mosquitto Broker*, assim como o uso da ferramenta de programação visual *Node-RED* e dos microcontroladores *ESP32* e *ESP8266*.

Em relação ao trabalho pode-se abstrair os seguintes dados:

1. A utilização do protocolo *MQTT*, por meio do *Broker Mosquitto* em conjunto com os microcontroladores da *ESPRESSIF*;
2. A utilização de diversos sensores, validando que a conexão e transferência de dados através do protocolo *MQTT*;
3. Uma abordagem para a organização dos tópicos *MQTT* (Tabela 5.1), definindo dispositivos de publicação (*publishers*) e de inscrição (*subscribes*);
4. A utilização do *Node-RED* como agente de interfaceamento do projeto (Figura 5.1).

Figura 5.4 – Visão geral do projeto.

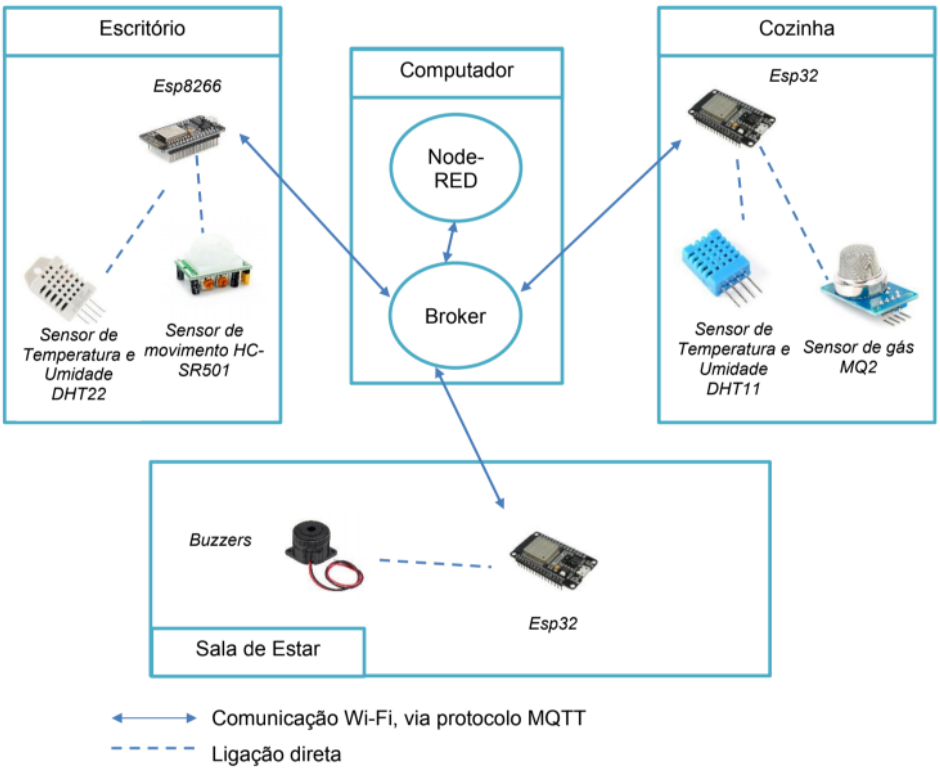


Figura 5.5 – Visualização da interface desenvolvida no *Node-RED* para receber entradas do usuário e mostrar dados dos sensores.

Tabela 5.1 – Organização dos tópicos *MQTT*.

Dispositivo	**Inscrição**	**Publicação**
ESP8266 (Escritório)	casa/escritório/esp8266/movimento	casa/escritório/esp8266/temperatura
		casa/escritório/esp8266/umidade
	casa/escritório/esp8266/tomada	casa/escritório/esp8266/movimento/estado
	casa/escritório/esp8266/luz	casa/escritório/esp8266/movimento/notificação
ESP32 (Cozinha)	casa/esp32/cozinha/fumaça	casa/cozinha/esp32/temperatura
	casa/cozinha/esp32/tomada	casa/cozinha/esp32/umidade
	casa/cozinha/esp32/luz	casa/cozinha/esp32/fumaça/estado
		casa/cozinha/esp32/fumaça/notificação
ESP32 (Sala de estar)	casa/escritório/esp8266/notificação	-
	casa/cozinha/esp32/notificação	

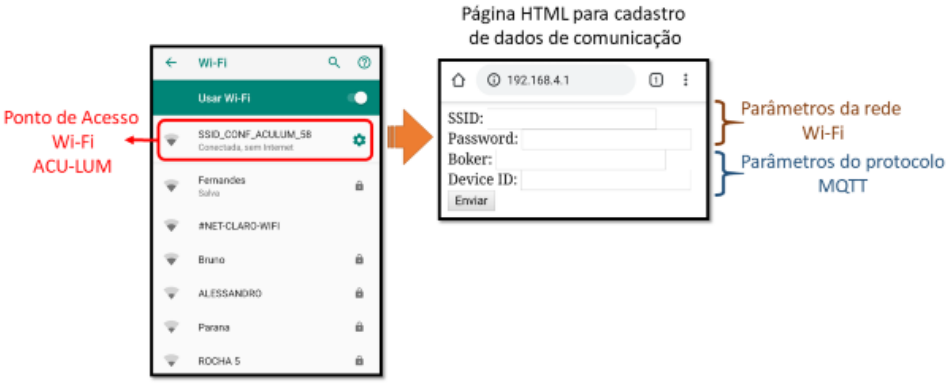
5.3 Desenvolvimento de plataformas embarcadas aplicadas a implementação de *Smart Buildings* com base no *framework SmartLVGrid*

Autor: Rubens de Andrade Fernandes

Este trabalho de conclusão de curso desenvolvido por Rubens de Andrade Fernandes pela Universidade do Estado do Amazonas concentrou-se no desenvolvimento de algorítimos para *software* embarcado e dispositivos de *hardware*, associados a plataformas microcontroladas e microprocessadas, com o objetivo de realizar a convergência *smart building* em sistemas de iluminação e medição de energia elétrica sem recursos de automação, comunicação e controle.

Os pontos relevantes deste trabalho em relação à para implementação deste TCC estão na criação de rotinas de cadastro geradas exclusivamente pelos microcontroladores *ESP32* utilizados (Figura 5.6).

Figura 5.6 – Configuração dos dispositivos



Essa rotina de cadastro faz com que o microcontrolador forneça um ponto de acesso local possibilitando qualquer o acesso de qualquer dispositivo. Ao realizar uma conexão através de um computador ou *smartphone*, por exemplo, o usuário pode informar dados de cadastro como para o dispositivo operar em rede.

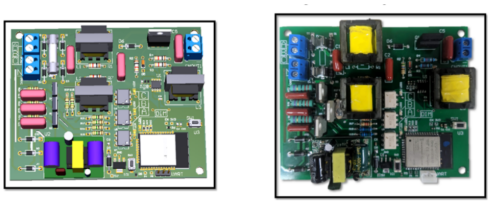
O segundo ponto que se pode destacar desse trabalho foi a criação de bibliotecas que auxiliam na operação do microcontrolador como demonstrado na tabela abaixo.

Tabela 5.2 – Bibliotecas implementadas para elaboração do *firmware*

Bibliotecas	Descrição
DriverAcuLum	Utilizada para implementar os métodos referentes as DRFs a serem executadas
DriverMQTT	Utilizada para garantir confiabilidade e segurança no uso do protocolo MQTT
DriverWIFI	Utilizada para garantir uma conexão mais estável, robusta e segura na rede Wi-Fi
SaveData	Utilizada para armazenar os parâmetros do ACU-LUM na EEPROM do ESP32

Outro ponto importante foi a confecção dos módulos propostos, utilizando técnicas de modelagem 3D para visualização *PCB's* e a realização de soldagem e inserção de componentes.

Figura 5.7 – Comparação entre o *layout 3D* e o dispositivo físico montado



5.4 Design, Implementation and Practical Evaluation of an IoT Home Automation System for Fog Computing Applications Based on *MQTT* and *ZigBee-WiFi* Sensor Nodes

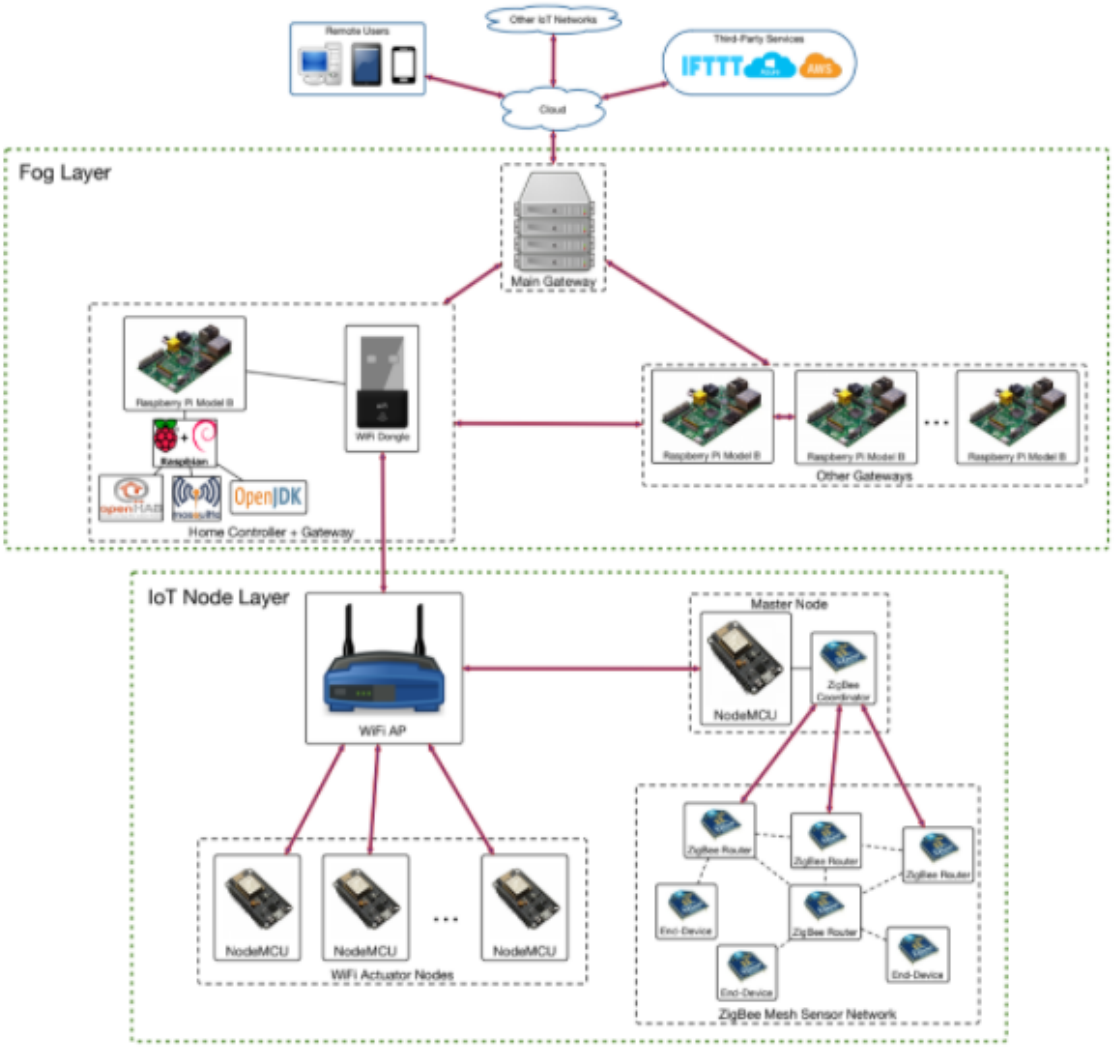
Autores: Iván Froiz-Míguez, Tiago M. Fernández- Camaramés, Paula Fraga-Lamas e Luis Castedo

Este artigo desenvolvido por membros do departamento de engenharia de computação, da Universidade da Coruña, Espanha, apresentou um estudo de caso e aplicação onde buscou-se unir duas tecnologias dentro das características da *IoT*: *WI-FI* e o protocolo *ZigBee*. O protocolo *ZigBee* é muito utilizado quando se tem a necessidade de uma comunicação à longas distâncias com um baixo consumo energético.

O trabalho apresentou a formulação de uma nova tecnologia: o *ZiWi*, um *HAS - Home Automation System* de computação em nuvem que preenche a lacuna entre dispositivos *ZigBee* e *Wi-Fi*, conectando sensores e atuadores perfeitamente para fazer uso de tais tecnologias em uma residência. O *ZiWi* utiliza o *Wi-Fi* para comunicação com atuadores já que, em geral, há a necessidade de estar continuamente operando e ouvindo comandos assíncronos. Enquanto o *ZigBee* é aplicado para sensores porque é ideal para enviar dados em intervalos periódicos, a fim de economizar energia (já que muitos sistemas dependem de baterias). Além disso, o sistema se concentra no crescente mercado de *IoT* e na utilização de sensores emergentes. Além disso, *ZiWi* faz uso da computação em nuvem, paradigma para fornecer conectividade entre o usuário e os diferentes eletrodomésticos, não apenas permitindo ao usuário controlá-los, mas também oferecendo a possibilidade dos dispositivos aprenderem com bancos de dados *online*. Isso é alcançado graças à natureza distribuída do *ZiWi*: o hardware do controlador doméstico é mantido com o mínimo de execuções de tarefas em tempo real, delegando o processamento dos dados e as decisões em tempo não real para servidores em nuvem remotos.

Com a utilização o *Wi-Fi*, o trabalho procurou integrar o protocolo *MQTT* (como a figura de um dispositivo *Broker*, a publicação e inscrição em tópicos) para dispositivos que trabalhavam com o protocolo *ZigBee*. A técnica desenvolvida se tornou muito interessante para aplicações em trabalhos posteriores a esse TCC, como a utilização em ambientes mais isolados.

Figura 5.8 – Esquema para aplicação do protocolo *ZiWi*



6 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão abordados os aspectos teóricos necessários para o melhor discernimento das definições e componentes a serem utilizados ao longo do projeto. De início, serão abordados os conceitos base como redes *Wireless*, *Internet of Things*, *Smart Things* e o protocolo *MQTT*. Conceitos como a montagem de circuitos eletrônicos, sensoriamento, implementação de sistemas embarcados assim como as tecnologias envolvidas na criação de aplicações *mobile* e *desktop* também serão tratadas neste capítulo. As subseções posteriores, agruparão os conceitos em elementos: de base, de *Hardware*, de *Firmware* e de *Software*.

6.1 Elementos Base

Esta seção discorre sobre as tecnologias e metodologias que serão utilizadas em todas as vertentes deste trabalho, destacando-se as partes mais relevantes.

6.1.1 Redes *Wireless*

O termo *Wireless* provém do inglês: *wire* (fio, cabo); less (sem); caracteriza qualquer tipo de conexão para transmissão de sem a utilização fios ou cabos (VENTAVOLI, 2014). É importante notar que uma rede sem fio possibilita um conjunto de equipamentos conectados através de sinais de rádio frequência possuindo vantagens cruciais para a aplicação do conceito de Internet das Coisas:

- **Maior produtividade -** disponibiliza acesso à rede em todo o raio de alcance onde o ponto de acesso está instalado, oferecendo liberdade de deslocamento com conexão contínua;
- **Flexibilidade de instalação -** podem ser instaladas em locais com temperaturas elevadas, em que os cabos não suportariam, ou em locais que necessitam de acesso temporário;
- **Redução de custo -** reduzem os custos de instalação, dispensando o uso de material para cada ponto de conexão;
- **Interoperabilidade e segurança -** capaz de comunicar sistemas de forma confiável e com segurança, possuindo chaves de acesso e até mesmo transferindo mensagens criptografadas.

Dentre as diversas características podemos destacar que as redes *wireless* são dividas quanto a **abrangência de sinal** em:

- ***Wireless Local Area Network* - WLAN -** rede de área local;
- ***Wireless Metropolitan Area Network* - WLAN -** rede de região metropolitana;
- ***Wireless Wide Area Network* - WWAN -** rede de área mundial sem fio;
- ***Wireless Local Loop* - WLL -** acesso fixo sem fio;
- ***Wireless Personal Area Network* - WPAN** - redes pessoais sem fio;

Fundado em 1987, o grupo de trabalho *Institute of Electrical and Electronics Engineers - IEEE* 802.11 *Wireless LAN*, realizou a padronização das *WLANs*. Desenvolveu-se o padrão DS-SS IEEE 802.11b, chamado de *Wi-Fi* pela *Wireless Ethernet Compatibility Alliance* - WECA o qual é amplamente usado no mercado atual. (DIAS, 2016). A partir dessa padronização é possível aplicar as transmissões de dados por diversos protocolos de maneira barata e prática.

A topologia de uma rede IEEE 802.11 (*Wi-Fi*) possui os seguintes elementos-chave:

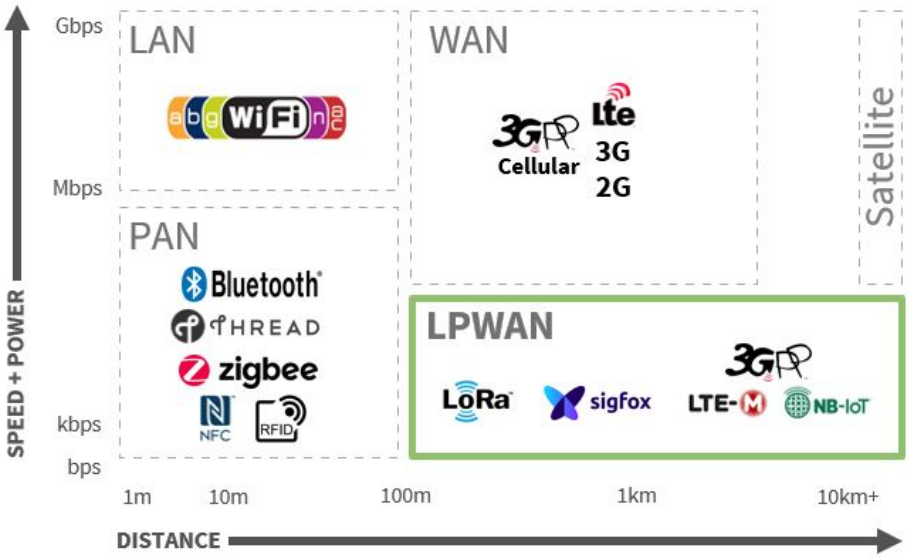
- **BSS - *Basic Service Set* -** corresponde a uma célula de comunicação *wireless*;
- **STA - *Stations* -** são as estações de trabalho que comunicam-se entre si dendo da **BSS**;
- **AP - *Access Point* -** coordena a comunicação entre as **STA's** dentro da **BSS**. Na maioria das vezes roteadores realizam tal operação;
- ***Bridge* -** faz a ligação entre diferentes redes, por exemplo, uma rede sem fio para uma rede cabeada convencional;
- **ESS - *Estended Service Set* -** consiste de várias células **BSS's** vizinhas que se interceptam e cujos **AP's** estão conectados a uma mesma rede tradicional. Nestas condições uma **STA** pode movimentar-se de um **BSS** para outra, permanecendo conectada à rede. Este processo é denominado *Roaming*.

6.1.2 A Internet das Coisas

Existe uma diversidade de ideias em relação ao conceito de Internet das Coisas (*Internet of Things - IoT*), sendo que todos os conceitos, em resumo, fazem alusão à um conjunto de tecnologias e protocolos associados que permitem que objetos se conectem a uma rede de comunicações e onde são identificados e controlados.

Esse termo se tornou possível graças aos avanços tecnológicos e as reduções de custo de todos os dispositivos eletroeletrônicos, os quais possuem a capacidade de comunicação principalmente por meio de protocolos *Wiress* (Figura 6.1). Com a aplicação em massa do *IoT* surge a *hiperconectividade*, termo que descreve a possibilidade da aquisição descomunal de informações as quais podem ser utilizadas para melhorar um processo, produto ou serviço.

Figura 6.1 – Protocolos utilizados para aplicacão do *IoT* com base nos conceitos de redes. Relação entre velocidade-força e distância.



Fonte: iot.do, 2021

6.1.3 O protocolo MQTT

O protocolo ***Message Queuing Telemetry Transport*** - MQTT foi inventado e desenvolvido inicialmente pela *International Business Machines* - IBM no final dos anos 90. Sua aplicação original era vincular sensores em *pipelines* de petróleo a satélites.

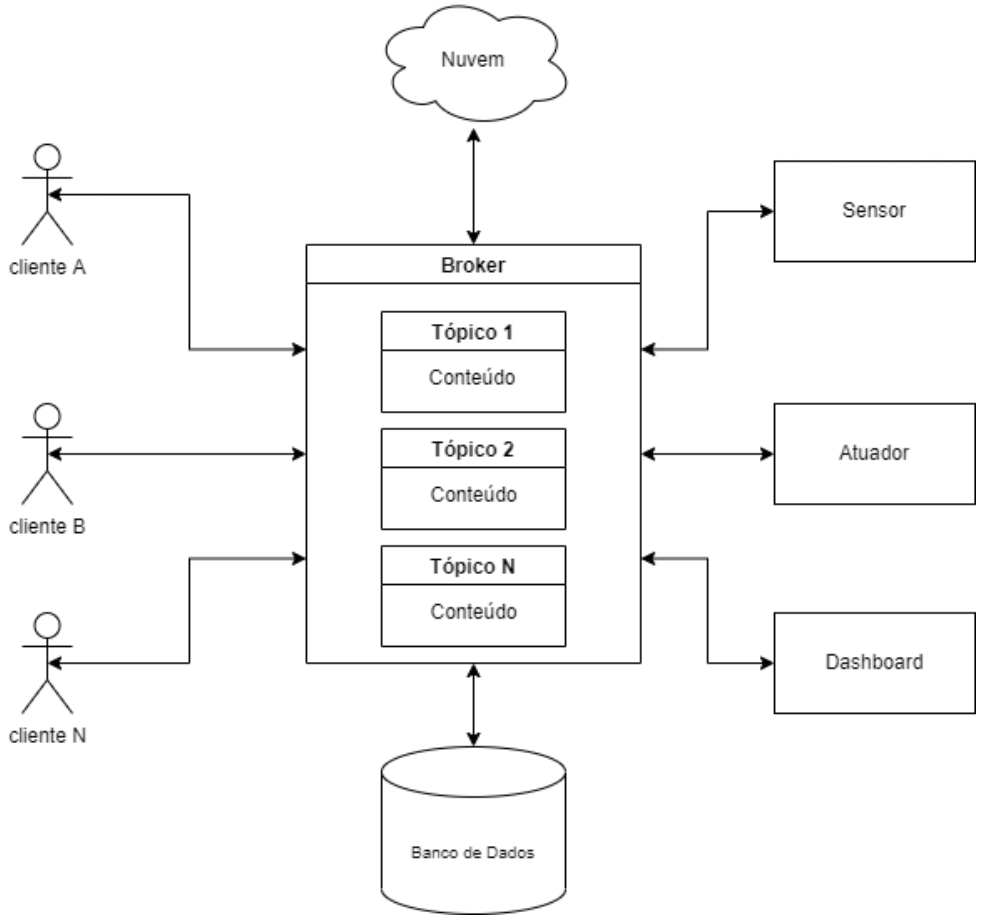
Como seu nome sugere, ele é um protocolo de mensagem com suporte para a comunicação **assíncrona** entre as partes. Um protocolo de sistema de mensagens assíncrono desacopla o emissor e o receptor da mensagem tanto no espaço quanto no tempo e, portanto, é escalável em ambientes de rede que não são confiáveis. Apesar de seu nome, ele não tem nada a ver com filas de mensagens, na verdade, ele usa um modelo de publicação e assinatura. No final de 2014, ele se tornou oficialmente um padrão aberto OASIS, com suporte nas linguagens de programação populares, usando diversas implementações de *software* livre (IBM, 2019).

O MQTT foi projetado para aplicações que utilizam pouca banda de rede, com requisitos de *hardware* extremamente simples e leve. Ele está na mesma camada OSI que o ***Hypertext Transfer Protocol*** - HTTP, porém a maior diferença entre eles é o tamanho do *payload*. No HTTP o *payload* [1] é maior, o que inviabiliza o uso em conexões de baixa qualidade. Além disso o MQTT possui maior segurança, apresenta mais níveis de serviço, é menos complexo e permite uma comunicação de 1 para N se comparado ao HTTP que também é um protocolo utilizado no universo de Internet das Coisas (UFRJ, 2019).

Em uma rede MQTT existem dois agentes principais: o ***broker*** e os ***clients***. O *broker* é um servidor que centraliza as mensagens dos clientes e as encaminha para os clientes interessados. Já o cliente é um dispositivo ou serviço que tenha capacidade de interagir com o *broker* e trocar mensagens. Podemos exemplificar como tipos de clientes (Figura 6.2) um sensor, um controlador, um atuador ou até mesmo um *software* sendo executado em alguma instância da rede.

[1] ***payload*** - Dado de interesse sem metadados, sem cabeçalho de transmissão ou outras informações acessórias usadas apenas como infraestrutura para transmitir o que importa.

Figura 6.2 – Diagrama básico MQTT



Fonte: própria, 2021

Como qualquer outro protocolo de comunicação o MQTT possui diversos termos e características associadas, como a segurança, a definição dos *endpoints*[2], o endereçamento, tipo de dado trafegado, entre outros. Abaixo estão pontuados os dados cruciais para a aplicação deste trabalho.

- ***broker***: é o intermediário no processo de comunicação, atuando como um servidor;
- ***client***: responsável por estabelecer e manter uma conexão com o *broker*, enviar e receber as mensagens;
- ***broker ip***: identificação única do servidor ***broker*** conectado em determinada rede;
- ***broker username e broker password***: credenciais, opcionais, para determinado cliente estabelecer conexão com o servidor;
- ***QoS***: nível de qualidade do serviço desejado, indicando como deve ser a relação entre os elementos comunicantes. *QoS 0*: Não há a confirmação de entrega de uma mensagem e quem a envia não tem obrigação de manter a mensagem armazenada para futuras retransmissões. *QoS 1*: Existe a confirmação de entrega da mensagem. As mensagens enviadas são armazenadas por quem as envia. *QoS 2*: Garante que a mensagem seja entregue exatamente uma vez, com envio de confirmações de recebimento e confirmações de recebimento de confirmações de recebimento. Enquanto uma mensagem não é confirmada, ela é mantida.(FABIOBRANDAO, 2019)

[2]**endpoints** - Pontos de extremidade. Terminais de conexão entre uma API e o cliente.

- ***last good message***: é a operação à qual o *broker* envia a última menagem válida recebida em um determinado tópico ao ser requisitado por um cliente. Normalmente, se um cliente MQTT assina um tópico em um *broker*, ele não receberá nenhuma das mensagens publicadas nele antes da assinatura. Se um cliente publicar uma mensagem em um tópico com o sinalizador de retenção definido como *True*, o corretor salvará essa mensagem como a "Última Mensagem válida" nesse tópico. Esta mensagem será recebida por qualquer cliente que assine esse tópico (MNTOLIA.COM, 2019).
- ***last will testament (LWT)***: são mensagens pré-definidas a serem publicadas pelo broker em nome de um determinado cliente, uma vez que esse cliente está *offline* e não pode publicar mais;
- ***keep alive***: mensagens periódicas enviadas por determinado cliente buscando validar a conexão;
- **tópicos, *publish* e *subscribe***: O ato de um cliente enviar uma mensagem é chamado *publish*(publicação). E para receber mensagens de determinado um tópico um cliente deve fazer um *subscribe*(inscrição). Os níveis de um tópico são separados por "/" e um cliente pode optar por se inscrever em quantos tópicos forem necessários, utilizando os artifícios da Tabela 6.1.

Símbolos	Descrição	Exemplo
+	Retorna ou envia qualquer informação naquele nível (Coringa)	area/10/sensor/5000/temperatura area/10/sensor/4000/temperatura area/10/sensor/+/temperatura
#	Retorna ou envia qualquer coisa abaixo daquele nível	area/10/#
$	Tópicos iniciados com $ são especiais usados internamente pelo broker.	$SYS/broker/clients/total

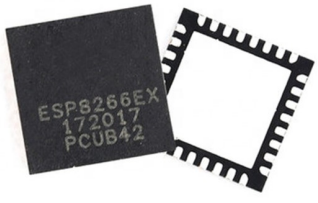
Tabela 6.1 – Caracteres especiais utilizados para envio e recebimento no protocolo MQTT.

6.2 Elementos de *Hardware*

6.2.1 O microcontrolador ESP8266

O *ESP8266* (Figura 6.3) é um *chip* microcontrolador da fabricante chinesa *Espressif Systems* construído em torno de um processador Tensilica Xtensa LX3, inclui *Wi-Fi on-board*. Originalmente concebido como um adaptador *UART* para *Wi-Fi* (utilizado em *tablets*), permitindo que outros microcontroladores se conectem a uma rede *Wi-Fi* e façam conexões *TCP/IP* simples usando comandos do estilo *Hayes*[3], o *ESP8266* rapidamente se tornou popular como um microcontrolador autônomo devido ao seu baixo custo de mercado.

Figura 6.3 – *ESP8266EX*.



Fonte: DIGIKEY, 2020

[3]**comandos *Hayes*:** série de *strings* curtas que podem ser combinadas para produzir comandos de operações como discar, desligar e alterar os parâmetros de conexão.

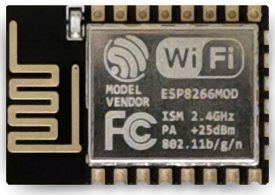
Apesar da falta de documentação inicial, uma grande comunidade foi formada em torno do ESP8266, integrando e dando suporte ao *firmware* para o chip, fazendo-o compatível com a plataforma *Arduino*. Embora o chip *ESP8266* seja feito pela *Espressif*, existem diversos módulos criados para aplicações distintas.

Os modelos podem se diferenciar pela quantidade de pinos extenados, memória *flash* instalada, modelo de antena, entre outros. Para a realização deste trabalho foram escolhidos os modelos: *ESP-12S* (Figura 6.4) e *ESP-12E* (Figura 6.5) presentes, respectivamentes nas placas de desenvolvimento WEMOS D1 e NodeMCU. Ambos possuem 11 *GPIOs - General Purpose Input Output* sendo ideais para a leitura de uma gama de sensores. A diferença entre ambos está no desenho da antena integrada, sendo a do ESP-12S mais otimizada e, na quantidade de pinos externados, o ESP-12S não extena os pinos voltados para conexão *SPI* e de cartões *SD* (ESP8266.NET, 2021). As características gerais e específicas para o densenvolvimento deste trabalho estão descritas no **anexo A**.

Figura 6.4 – Vista superior do *ESP-12S*.

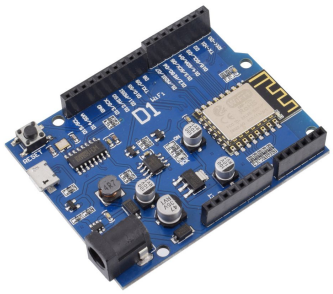
Fonte: FILIPEFLOP, 2020

Figura 6.5 – Vista superior do *ESP-12E*.



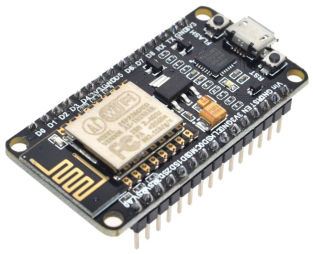
Fonte: FILIPEFLOP, 2020

Figura 6.6 – Wemos D1 com ESP-12S.



Fonte: esp8266.org, 2021

Figura 6.7 – Nodemcu com ESP-12E.



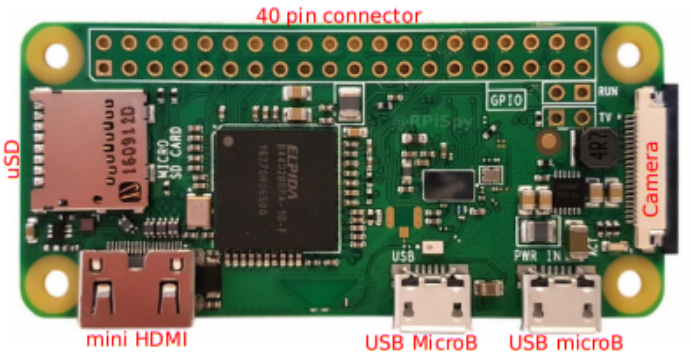
Fonte: esp8266.org, 2021

6.2.2 A Raspberry Pi Zero W

A *Raspberry Pi Zero W* é basicamente um computador de placa única. Ela possui recursos como *slot* para cartão *microSD*, conectores *HDMI* e de câmera, conectividade *Wi-Fi* e *Bluetooth 4.0*, um conector macho de entrada-saída (*GPIO*) de 40 pinos e mini conector de alimentação *+ 5VDC*. O microcontrolador baseado no processador *BCM2835 ARMv7 system-on-chip (SoC)* alimenta o Pi Zero W.

Essa versão de *hardware* da *Raspberry Pi Foundation* é muito compacta, ficando no meio termo entre as fazes de prototipação e aplicação. Com o microprocessador *ARM BCM2835* de *1GHz*, memória *RAM* de *512MB* e um sistema operacional enxuto instalado, a *Raspberry Pi Zero W* é ideal para rodar aplicações como um *Broker MQTT*. Informações adicionais sobre esse dispositivo estão disponíveis no **Anexo B**.

Figura 6.8 – Vista superior da *Raspberry Pi Zero W*

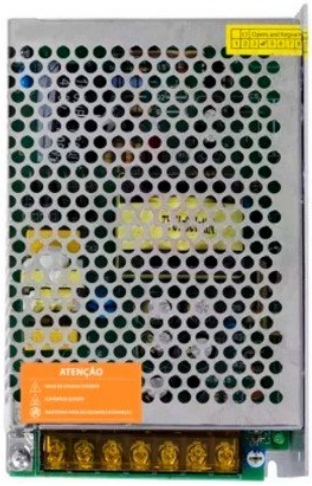


Fonte: SPARKFUN, 2020

6.2.3 As fontes de alimentação DC

Para alimentação elétrica de todos os dispositivos eletrônicos envolvidos no processo de automação da cisterna, foram utilizadas dois tipos de fontes chaveadas: de *12V/20A* para o módulo **CCM** e *12V/10A* para o módulo **TCM**, representadas pela Figura 6.9. Essas fontes são ideais por possuir características robustas, como proteção eletromecânica e potência necessária para ativação de motores e válvulas solenoides. Há também a disponibilização de vários conectores na saída, os quais podem ser interligados para pontos específicos de alimentação.

Figura 6.9 – Vista superior da fonte



Fonte: AMAZON, 2020

6.2.4 A motobomba ou bomba d'água

A bomba d'água selecionada, Figura 6.10 , trabalha com fontes de alimentação de correntes contínuas (*DC*) o que permite um maior controle do funcionamento através de sistemas *microprocessados*. Essa bomba opera a *12V*, com potência máxima de *80W*. Ela possui a capacidade de exercer uma vazão de *5,5L/min* com ganho de elevação de no máximo $40m$.

Figura 6.10 – Bomba d'água *DC*

Fonte: ENERGIA TOTAL, 2020

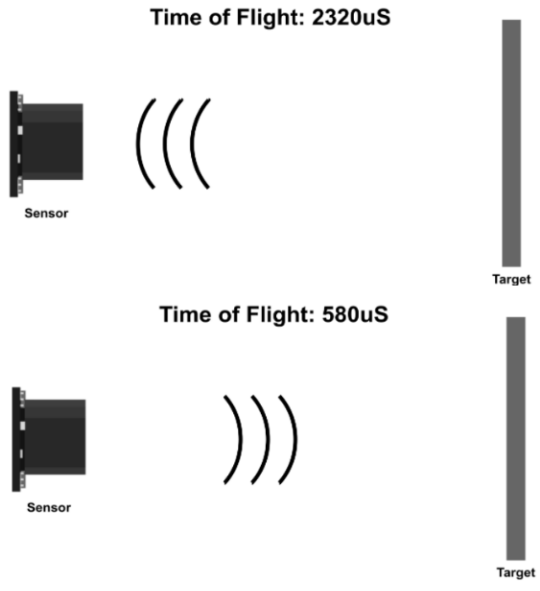
6.2.5 O sensor ultrassônico

Um sensor ultrassônico é um instrumento que mede a distância até um objeto usando ondas sonoras de frequência acima do alcance da audição humana (ultrassônicas). Ele usa um transdutor para enviar e receber pulsos ultrassônicos os quais são utilizados para medição de distância através de lapsos temporais (Figura 6.11). A equação de distância associada é dada por:

$$d = \frac{\Delta t \cdot v}{2} \tag{6.1}$$

- d é a distância;
- Δt é a diferença entre o tempo de emissão e chegada da onda;
- v é a velocidade do som (tomada como $340 m/s$).

Figura 6.11 – Representação do funcionamento de um sensor ultrassônico



Fonte: Maxbotix, 2021

Para medição de nível selecionou-se o módulo ultrassônico *JSN-SR04T*. Esse módulo oferce uma região de medição de distância entre $20cm$ a $600cm$, com precisão na faixa de $2mm$.

Esse módulo de apenas quatro pinos: *5V* (VCC), *Trig (RX)*, *Echo (TX)* e *GND*, inclui um transceptor integrado além do seu circuito de controle o qual possui três modos de operação sendo selecionados pelo valor do resistor (R27) soldado a placa:

- **Modo de operação 1 (R27 não soldado)**: neste modo, o microcontrolador na placa não é utilizado. As operações de leitura ficam por responsabilidade externa.
- **Modo de operação 2 (R27 = $47k\Omega$)**: neste modo, o microcontrolador da placa entra em modo de leitura constante e envia dados a cada $100ms$. Os dados são enviados via o pino TX respeitando o protocolo: 0XFF + HIGH DATA + LOW DATA + SUM 1.
- **Modo de operação 3 (R27 = $120k\Omega$)**: neste modo, o microcontrolador entra em modo de baixo consumo e só envia os dados de leituras ao ser acionado pelo pino RX. Se a mensagem de acionamento for 0X55, a leitura de distância é enviada pelo TX da mesma forma do modo anterior.

Sobre o protocolo utilizado, cada parte da mensagem tem a seguinte definição:

- 0xFF: indica a chegada de uma mensagem;
- HIGH DATA: 8 bits mais significativos da leitura;
- LOW DATA: 8 bits menos significativos da leitura;
- SUM 1: O resultado da soma de todos os caracteres (0XFF + HIGH DATA + LOW DATA) para que a mensagem seja validada.

Figura 6.12 – Sensor ultrassônico JSN-SR04T.

Fonte: USINAINFO, 2020

O transceptor, resistente à água e poeira, é conectado ao módulo por meio de um cabo com *2,5 metros* de comprimento.

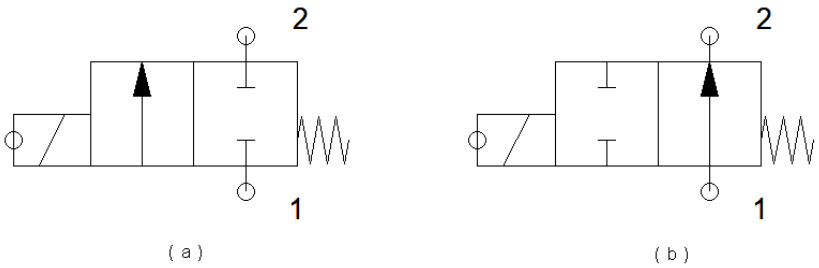
6.2.6 A válvula solenoide

Uma válvula solenoide (Figura 6.14) é um tipo de válvula que pode ser acionada ou desacionada eletronicamente. Seu componente principal é uma bobina elétrica com um núcleo ferromagnético móvel no centro chamado de êmbolo.

Dependendo da aplicação, uma válvula pode ser classifica em **Normalmente Aberta - NA** ou **Normalmente Fechada - NF**. Para uma **NF**, em sua posição de repouso, o êmbolo tampa um pequeno orifício por onde é capaz de circular um fluído. Quando uma

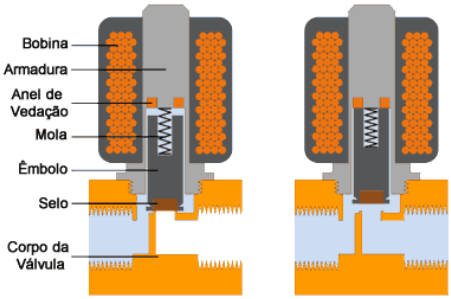
corrente elétrica circula na bobina, cria um campo magnético o qual exerce força sobre o êmbolo. Ele é deslocado em direção ao centro da bobina, abrindo o orifício e possibilitando a passagem do fluído. A operação para uma válvula **NA** é justamente o oposto, o êmbolo fica no centro e é deslocado com a formação do campo magnético. Outra classificação é quanto ao número de vias e as formas de controle. Para este trabalho nos atentaremos para válvulas de duas vias com retorno por mola, representadas na figura abaixo.

Figura 6.13 – Representação de válvulas: *(a)* Normalmente Fechada e *(b)* Normalmente Aberta. Ambas de duas vias.



Fonte: Própria, 2021

Figura 6.14 – Representação interna de uma válvula solenoide com seus principais componentes



Fonte: Citisystems, 2021

Alguns exemplos do uso de válvulas solenoides incluem sistemas de aquecimento, tecnologia de ar comprimido, automação industrial, piscinas, sistemas de aspersão, máquinas de lavar roupa, equipamentos odontológicos, sistemas de lavagem de carros e sistemas de irrigação (CITISYSTEMS, 2017).

O modelo de válvula solenoide utilizada foi da marca Nascimetal (Figura 6.15) a qual possui características como: *1/2"* x *1/2"*, tensão de operação em torno de $12V\ DC$, consumo médio, quando ativada, de $500mA$, pressão de operação à $8{,}0kgf/cm^2$ com vazão máxima de $40L/min$ e vida útil de 50 mil operações. Garantiu o controle do fluxo de água de acordo com os dados elétricos enviados pelo microcontrolador.

Figura 6.15 – Válvula solenoide

Fonte: Nascimetal, 2021

6.2.7 A chave de nível tipo boia

Para questões de segurança a chave de nível *RF-0H21D* foi utilizada para fomentar a redundância do sistema, caso aconteçam falhas na medição do sensor citado na seção 6.2.5. Essa chave indicará para o microcontrolador, por meio de contato elétrico, o momento exato para desativação da bomba d'água, evitando possíveis vazamentos no sistema acolado à caixa d'água (**TCM**).

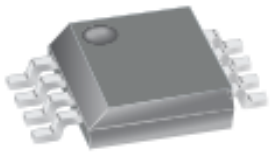
Figura 6.16 – Chave de nível *RF-0H21D*

Fonte: SMART KITS, 2020

6.2.8 O sensor de corrente

O sensor de corrente ACS712 selecionado é acessível e oferece precisão em soluções para detecção de corrente AC ou DC na indústria, comercial e sistemas de comunicação. O dispositivo permite fácil implementação: aplicações típicas incluem controle de motor, detecção de carga e gerenciamento, fontes de alimentação comutadas e sobrecorrente proteção contra falhas (MICROSYSTEMS, 2020).

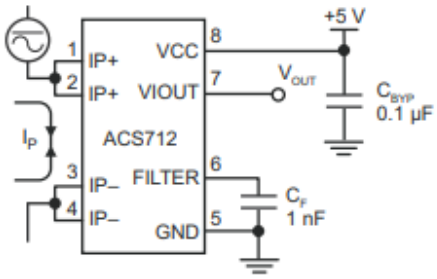
Figura 6.17 – Representação gráfica do sensor *ACS712*



Fonte: ALEGRO, 2020

O dispositivo consiste em um operador *Hall* linear preciso, a corrente aplicada flui através do material de cobre gerando um campo magnético que é detectado pelo *Hall* integrado e convertido em um proporcional de tensão. A precisão do dispositivo é otimizada por meio do fechamento proximidade do sinal magnético ao transdutor *Hall*. A resistência interna do material condutor atinge uma média de *1,2m*Ω, proporcionando baixa potência. De acordo *datasheet* do componente, pode-se observar a simples aplicação típica representada na figura abaixo.

Figura 6.18 – Aplicação típica do*ACS712*



Fonte: ALEGRO, 2020

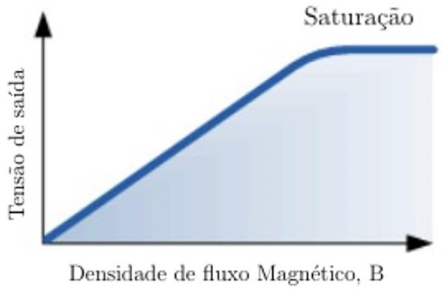
6.2.8.1 O sensor *Hall*

O efeito *Hall* é um fenômeno que pode ser observado quando um dispositivo condutor ou semicondutor se aproxima um campo magnético perpendicular. Nessa situação uma diferença de potencial pode ser medida através dos terminais de tal dispositivo. Essa queda de tensão está relacionada com a intensidade da corrente que gerou o campo magnético (ELECTRONICSTUTORIALS, 2021). A equação mostra essa relação:

$$V_H = R_H \left(\frac{I \cdot B}{t} \right) \tag{6.2}$$

- V_H é a tensão em Volts;
- R_H é o coeficiente do efeito *Hall*;
- I é o fluxo de corrente em Amperes;
- t é espessura do sensor em milímetros;
- B é a densidade do fluxo magnético em teslas;

Figura 6.19 – Comportamento da tensão de saída em relacão ao fluxo magnético de um sensor *Hall*.



Fonte: Eletronics Tutorials, 2021

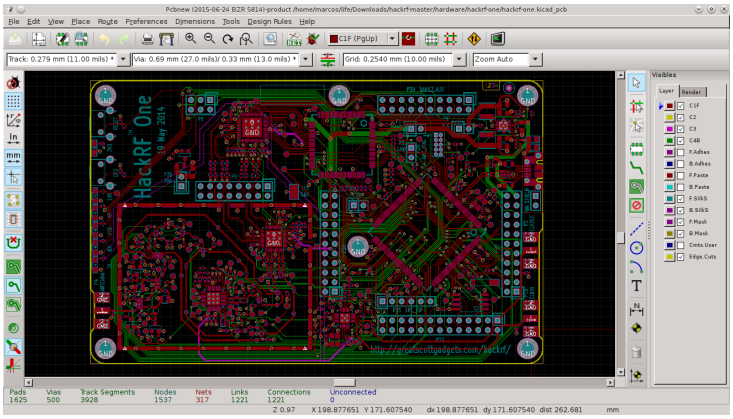
Em relação ao sensoriamento, os dispositivos *Hall* podem ser digitais ou analógicos. Os sensores analógicos fornecem uma saída de tensão contínua que de acordo com a intensidade do campo magnético e em alguns casos, como o ACS712, podem apresentar uma variação de saída linear (Figura 6.19). Conforme a intensidade do campo magnético aumenta, o sinal de saída do amplificador também aumentará até que comece a saturar pelos limites impostos a ele pela fonte de alimentação do circuito integrado. Essa saída linear pode ser conectada a um pino de leitura analógica de um microcontrolador ou microprocessador.

6.2.9 KiCad

O KiCad é um *Software* de código aberto para projetar circuitos eletrônicos. Essa ferramenta EDA - *Electronic Design Automation* lida com captura esquemática e layout de placas de circuito impresso (PCB - *Printed Circuit Board*) gerando arquivos *Gerber* (padrão universal de arquivo composto de uma combinação de comandos gráficos utilizados por equipamentos tipo *fotoploter* para a formação das imagens da placa de circuito impresso).

O *software* roda em ambientes *Windows*, *Linux* e *OS X* estando licenciado pela GNU GPL v3. Por meio de sua vasta biblioteca é possível construir esquemáticos de maneira prática. O KiCad também possui recursos de criação e adição de componentes, além de simulação via linguagem SPICE (ainda em beta) (KICAD, 2021).

Figura 6.20 – Exemplo de *PCB Design* utilizando *Kicad*



Fonte: kicad.org, 2021

6.3 Elementos de *Firmware*

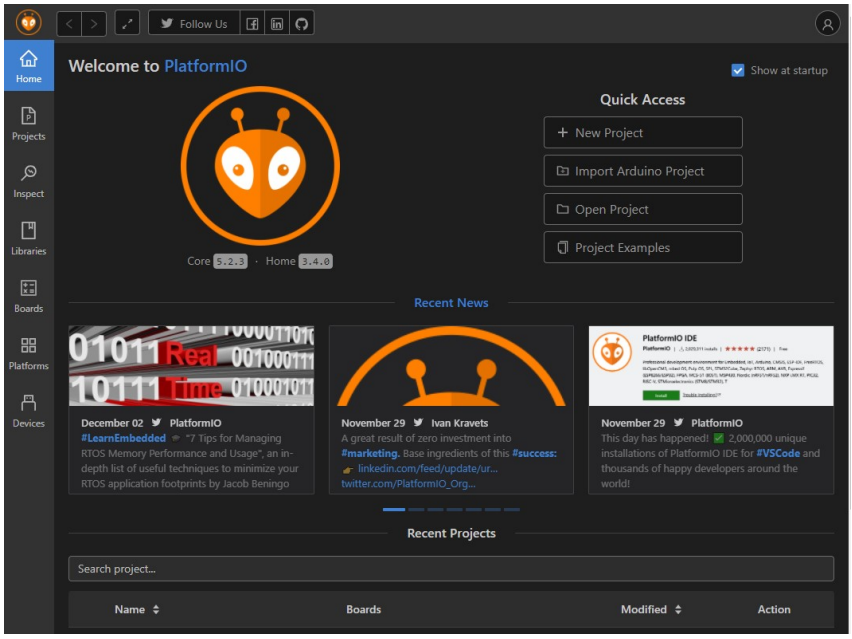
6.3.1 O Ambiente de Desenvolvimento *PlatformIO*

O *PlatformIO* é uma ferramenta profissional de plataforma cruzada, ou seja, disponível em diversas plataformas de desenvolvimento, e de estrutura múltipla voltada para desenvolvedores de *software* embarcado.

A ferramenta (Figura 6.21) fornece um ambiente de desenvolvimento integrado moderno, oferecendo suporte a diversos *kits* de desenvolvimento de *software* (*SDKs*) ou *frameworks* e inclui depuração sofisticada (*Debug Mode*), testes de unidade (*Unit Test*), análise automatizada de código (*Static Code Analysis*) e gerenciamento remoto (*Remote Development*). É arquitetado para maximizar a flexibilidade e escolha dos desenvolvedores, que podem usar editores gráficos e/ou de linha de comando (*PlatformIO Core*(*CLI*)).

Com esta interface, será possível a programação do microcontrolador *ESP8266* através da linguagem *C++* em conjunto com todas as bibliotecas disponibilizadas pela plataforma Arduino. Todas as funcionalidades, como rotinas de conexão *HTTP* e/ou *MQTT* através do *Wi-Fi*, Interrupção, *Timers* e Modos de Energia são completamente acessíveis.

Figura 6.21 – Extensão do *PlatformIO* rodando no *Visual Studio Code*



Fonte: Própria, 2021

6.3.2 *Linux* embarcado e a distribuição *DietPi*

Linux é um sistema operacional de código aberto amplamente utilizado em vários sistemas, possuindo compatibilidade com diversos tipos de microprocessadores, sejam eles de arquitetura *Reduced Instruction Set Computer* - RISC ou *Complex Instruction Set Computer* - CISC. Em todos os casos executa rotinas por baixo de todos os outros *softwares*, gerenciando todas as ações em conjunto com *hardware*.

O termo "*Linux*" tecnicamente se refere apenas ao *kernel* do *Linux*. A maioria das pessoas se refere a todo o sistema operacional como *"Linux"* porque, para a maioria dos usuários, um sistema operacional inclui um pacote de programas, ferramentas e serviços (como desktop, relógio, menu de aplicativo e assim por diante). Algumas pessoas, particularmente membros da *Free Software Foundation* , referem-se a esta coleção como *GNU / Linux* , porque muitas ferramentas vitais incluídas são componentes *GNU*. No entanto, nem todas as distribuições do *Linux* usam componentes *GNU* como parte do sistema operacional: o *Android* , por exemplo, usa um *kernel Linux*, mas depende muito pouco das ferramentas *GNU* (OPENSOURCE, 2021).

O mesmo *Linux* que roda em um supercomputador pode rodar também em uma simples placa eletrônica. O que torna isso possível é que o *Linux* suporta uma grande variedade de arquiteturas e processadores (SOFTWARELIVRE, 2021). Baseado na distribuição em Debian, conhecida por sua alta confiabilidade, a distribuição **DietPi**, utilizada neste trabalho, é extremamente leve, sendo altamente otimizada para o uso mínimo de recursos de CPU e RAM, garantindo que a *Central Process Unit* - CPU sempre opere em seu potencial máximo. Essa distribuição é suportada por diversos microprocessadores, incluindo o Broadcom BCM2835, presente na Raspberry PI Zero W.

Figura 6.22 – Catálogo com algumas de placas que suportam a distribuição DietPi.

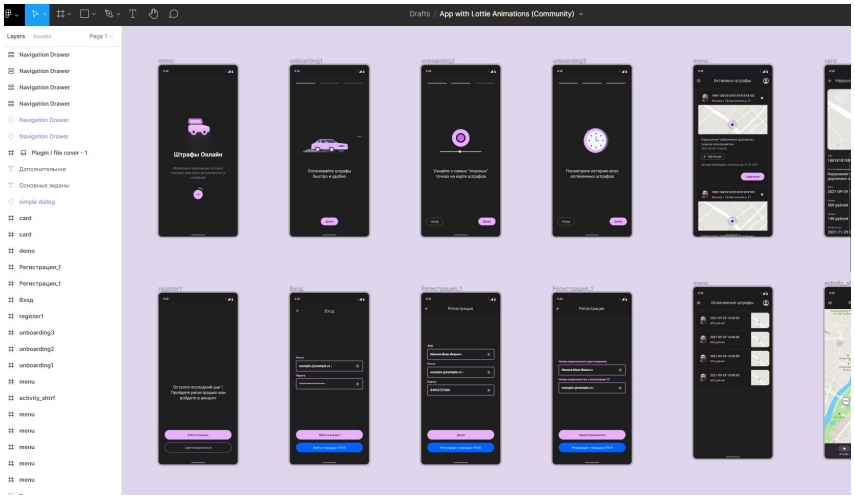
Fonte: dietpi.com, 2021

6.4 Elementos de *Software*

6.4.1 Figma

Figma é um editor gráfico de prototipagem para projetos de *design* baseado principalmente no navegador web, com ferramentas *offline* adicionais para aplicações *desktop*, *GNU/Linux*, *macOS* e *Windows*. Possui também a ferramenta *Figma Mirror*: um sistema de prototipagem que espelha o que está sendo feito no computador para o *smartphone Android* e/ou *iOS*, permitindo a simulação do desenho criado, como um aplicativo ou página da web. Por meio desse *software* é possível criar e manipular animações e transições de telas, focando no desenvolvimento de *User Interface Experience* - UI/UX (WIRED.COM, 2021).

Figura 6.23 – Exemplo de um projeto desenvolvido no Figma.



Fonte: figma.com, 2021

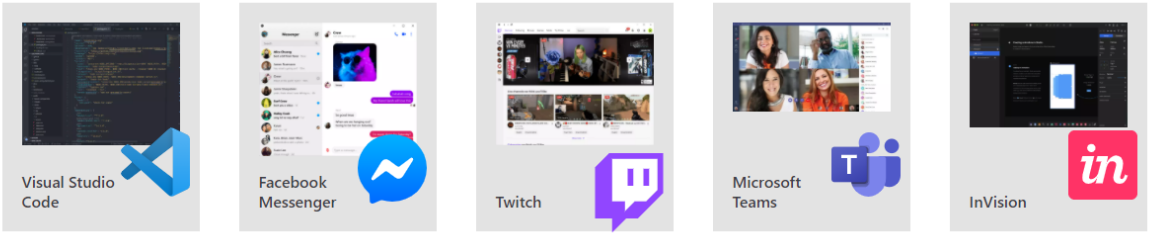
6.4.2 *Node.js*

O *Node.js* é um ambiente de execução (*runtime*) *Javascript* possibilitando criar aplicações sem depender de um *browser* para a execução. Por meio desta tecnologia, se torna possível empregar o *Javascript* para executar comandos em terminal assim como utilizá-lo para programar qualquer tipo de aplicação *backend* com auxílio de inúmeras bibliotecas desenvolvidas. As bibliotecas do *Node.js* podem ser acessadas e instaladas por seu gerenciador de pacote oficial, o *Node Package Manager* - **NPM** (UMBLER.COM, 2021).

6.4.3 *Electron*

O *Electron* é um *framework* multiplataforma (Windows, Linux e MacOS) para criação de interfaces possibilitando o usuário acessar serviços do sistema operacional tanto via linha de comando - *CLI* e interface gráfica - *GUI* (ELECTRONJS.ORG, 2021).

Figura 6.24 – Aplicações que utilizam *Electron*



Fonte: Adaptado (electronjs.org, 2020)

Por meio dele podemos desenvolver aplicações *desktop* utilizando *HTML*, *CSS*, *Javascript* em conjunto com todas as ferramentas aplicadas a um desenvolvimento *frontend web*. Incorporando o *Chromium*, navegador de código aberto (CHROMIUM.ORG, 2021), e *Node.js* no seu binário, toda a base de código fica por conta do *JavaScript* eliminando a experiência de desenvolvimento nativo.

Uma aplicação *Electron* tem como base os seguintes arquivos:

- ***package.json***: Aponta para o arquivo principal do aplicativo e lista seus detalhes e dependências;
- ***main.js***: Inicia a aplicação e cria a janela de visualização possibilitando diversas configurações;
- ***index.html***: O ponto de partida para a renderização.

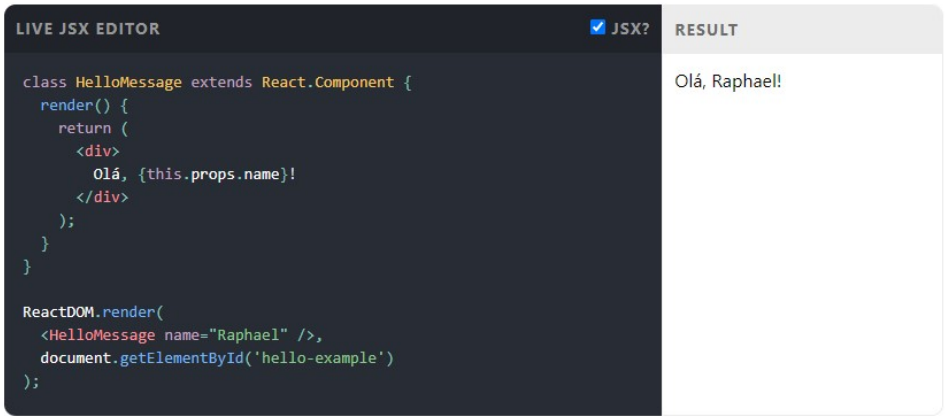
6.4.4 *React*

O *React* é uma biblioteca *JavaScript*, desenvolvida pela equipe do *Facebook*, que possui ferramentas que facilitam a construção de Interfaces na *Web*. Ele transforma um ambiente complexo, cheio de *Edge Cases* (casos onde você precisa tratar eventos e como os dados são manipulados) em algo muito mais simplificado (KENZIE.COM.BR, 2021).

Trabalhando no contexto declarativo, essa biblioteca permite criar telas para cada estado da aplicação, atualizando e renderizando apenas os itens necessários de forma eficiente. Os itens podem ser componentes (arquivos *JSX*) encapsulados que gerenciam seu próprio estado.

JSX é uma forma de criar elementos para serem utilizadas como *templates* de aplicações *React*. São bem similares ao código *HTML*, porém, não é interpretado pelo navegador. Por este motivo, deve-se utilizar um transpilador [4] para essa conversão. Atualmente, o mais conhecido deles é o *Babel*.

Figura 6.25 – Exemplo de componente *React*



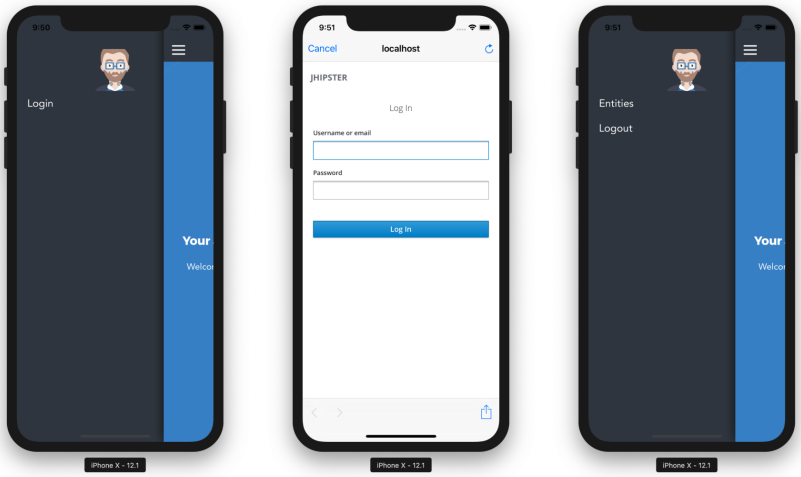
Fonte: Adaptado (reactjs.org, 2021)

6.4.5 *React Native*

Baseado no *React*, para desenvolvimento *WEB*, o *React Native* é um *framework* de codigo aberto desenvolvido pela equipe do *Facebook* que suporta a criação de aplicativos *mobile* multiplataforma (Android e iOS), sem que haja a preocupação de lidar com as linguagens padrões como *Java* ou *Swift*, usando apenas *Javascript*. Como ponto positivo, ao contrário de outros *frameworks* com o mesmo propósito, todo código desenvolvido com *React Native* será convertido para a linguagem nativa do sistema operacional, o que torna a aplicação mais rápida.

[4] **transpilador**: subconjunto de compiladores os quais identificam um arquivo de código-fonte e o convertem em outro arquivo de código-fonte, sendo de outra linguagem ou em uma versão diferente da mesma linguagem.

Figura 6.26 – Exemplos de *Templates* produzidos com *React Native*



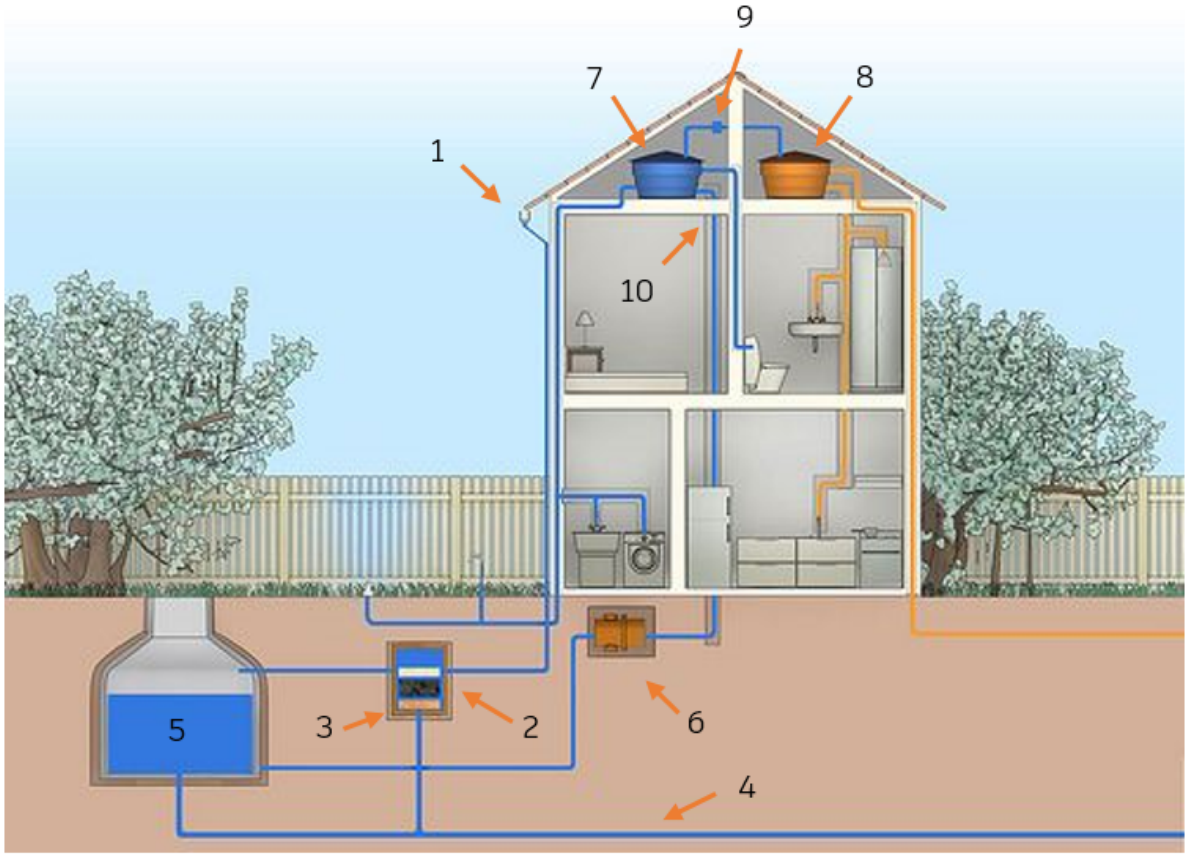
Fonte: Adaptado (developer.oka, 2021)

7 METODOLOGIA

7.1 Análise e definição das características do projeto

Para a metodologia deste trabalho de conclusão de curso, partiu-se do princípio de uma estrutura previamente construída, demostrada no esquema abaixo. A partir dos elementos dessa figura, evidenciados na Tabela 7.1, foi possível realizar uma análise sobre o que pode ser automatizado, levando em consideração as variáveis e características do sistema.

Figura 7.1 – Esquema de demonstração de uma cisterna no subsolo



Fonte: Adaptado (ECOMONTES, 2016)

Identificador	Elemento	Descrição
1	Calha coletora	Elemento convencional para coleta e descarte de água da chuva
2	Filtro A (cascalho fino)	Elemento para realização de filtragem de pequenas impurezas
3	Filtro B (cascalho grosso)	Elemento para realização de filtragem de impurezas
4	Tubulação de descarte	Tubulação utilizada como rota de escoamento quando o reservatório não está em uso ou está cheio
5	Reservatório de coleta	Cisterna propriamente dita
6	Motobomba ou bomda d'água	Elemento utilizado para realização do ganho de elevação da água
7	Caixa d'água auxiliar	Caixa d'água convencional com alimentação oriunda da bomba d'água
8	Caixa d'água convencional	Caixa d'água padrão com alimentação da estação de água da cidade
9	Elo de ligação	Ligação utilizada para abastecer a caixa d'água auxiliar quando a cisterna está seca ou em manutenção
10	Distribuidor	Elementos de distribuição de água para pontos estratégicos

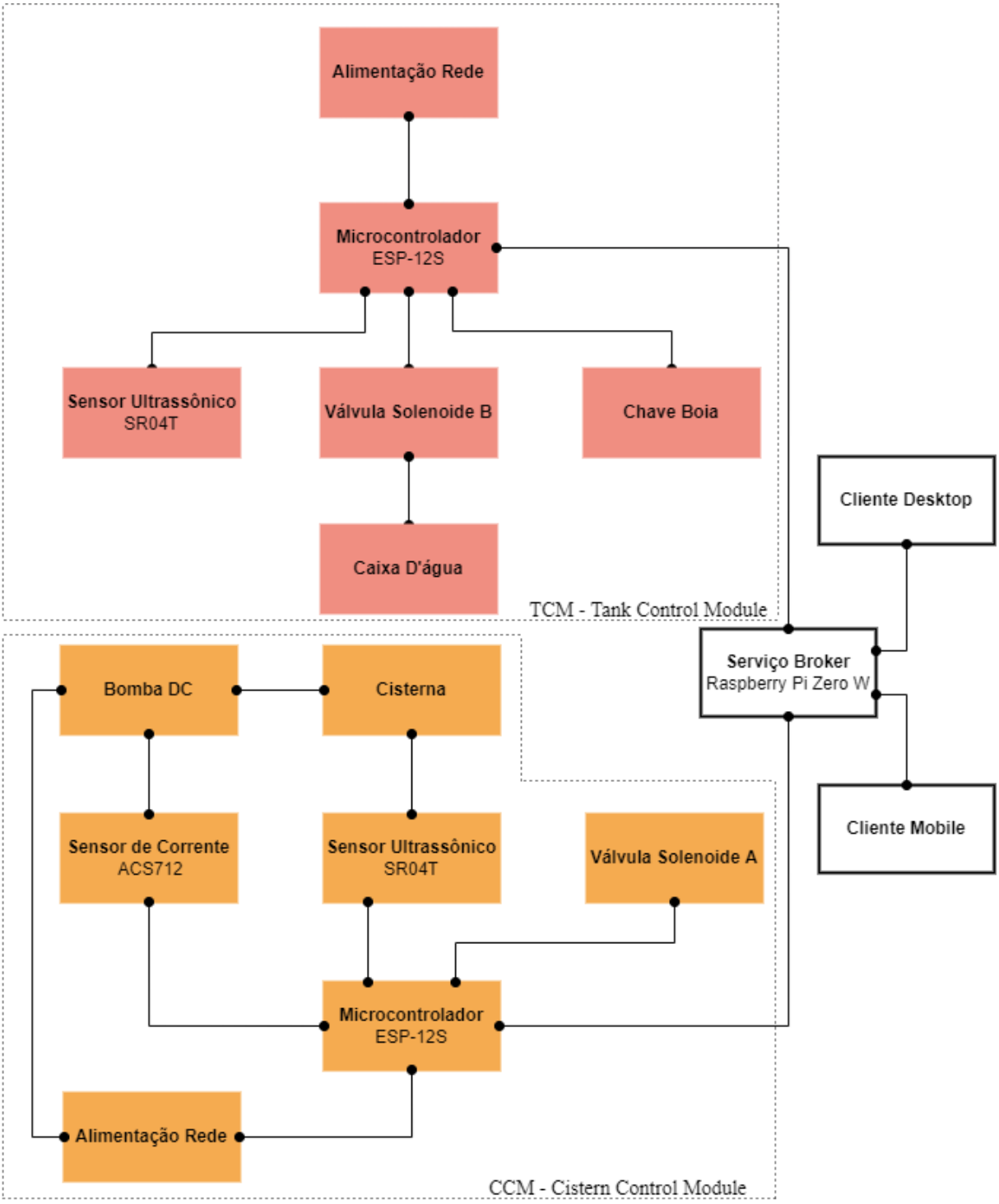
Tabela 7.1 – Identificação dos elementos da Figura 7.1.

Com base nesses elementos é importante destacar os seguintes pontos para as implementações que serão tratadas nas seções posteriores:

- Definir o método ou sistema para medição de nível do item 5;
- Elaborar o sistema de ativação/desativação da motobomba do item 6;
- Definir o método ou sistema para medição de nível do item 7;
- Elaborar um sistema para intermediar/controlar a passagem de água no elo de ligação (item 9);
- Definir um sistema para administrar o descarte de água da cisterna (item 4);
- Esquematizar e situar os sistemas para envio e aquisição de dados, tornando-se possível realizar remotamente as operações citadas nos itens anteriores;
- Configurar um pequeno servidor para intermediar entre os dispositivos microcontrolados e as interfaces de controle;
- Implementar as interfaces para visualizar/comandar os pontos citados nos itens anteriores.

Dois módulos distintos foram propostos: ***Tank Control Module* - TCM**, o qual será responsável pelo monitoramento e controle da caixa d'água auxiliar (Figura 7.1, identificador 7) e ***Cistern Control Module* - CCM**, responsável pelo monitoramento e controle da cisterna (Figura 7.1, identificador 5). O diagrama da Figura 7.2 mostra o esquema proposto para a criação do projeto.

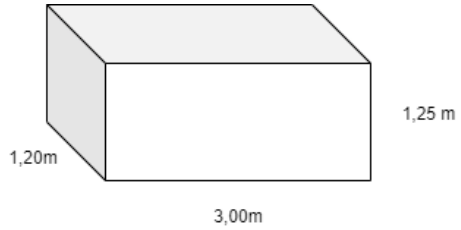
Figura 7.2 – Esquema básico do projeto.



Fonte: Própria

Outro ponto importante desta parte do projeto foi estabelecer as medidas de volume da cisterna e da caixa d'água assim como as equações de volume tendo como variável a altura do líquido. Para a cisterna, considerou-se um reservatório no formado de um paralelepípedo retângulo, com as dimensões definidas:

Figura 7.3 – Representação das dimensões da cisterna.



Fonte: Própria

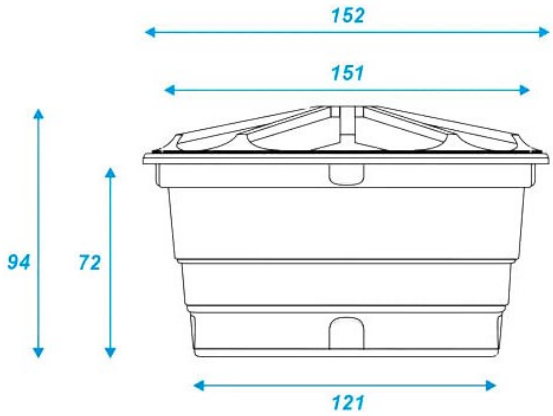
Equação de volume associada:

$$V = c \cdot l \cdot (h - s) \tag{7.1}$$

- V é o volume;
- h é a altura do tanque;
- s é a leitura de distância do sensor;
- c é o comprimento do tanque;
- l é o largura do tanque.

Para a caixa d'água, considerou-se um reservatório de 1000L, no formato de tronco de cone (formato padrão):

Figura 7.4 – Representação das dimensões da caixa d'água (valores em centímetros).



Fonte: tendtudo.com, 2021

Equação de volume associada:

$$V = \frac{\pi \cdot (h - s) \cdot (R^2 + R \cdot r + r^2)}{3} \tag{7.2}$$

- V é o volume;
- h é a altura do tanque;
- s é a leitura de distância do sensor;
- R é o raio maior;
- r é o raio menor.

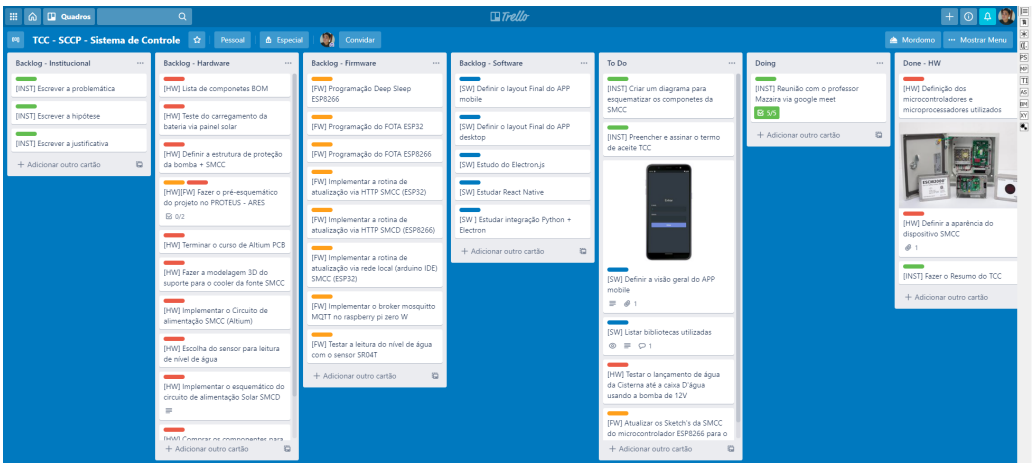
7.2 Organização do trabalho

Iniciou-se com a criação do quadro *Kanban* (Figura 7.5), para auxílio da organização das tarefas, e com a criação do repositório no *Github* (Figura 7.6) para armazenamento e versionamento dos códigos das vertentes de *Firmware* e *Software* do projeto. Deu-se prosseguimento com os estudos e levantamentos bibliográficos relacionando três áreas do projeto. Buscou-se encontrar os métodos, ferramentas, tecnologias, bibliotecas e *frameworks* mais adequados para a implementação do projeto. Diante de cada seleção feita, foram executadas análises para que fosse definido o funcionamento do sistema total, munido da combinação de cada uma das três áreas citadas anteriormente e visando a geração de um produto que pudesse ser empregado no mercado: atrativo economicamente e seguindo diretrizes sustentáveis.

Primeiramente, na seção de *hardware*, foram selecionados as ferramentas criação e simulação de circuitos, a escolha de todos os dispositivos eletrônicos a serem utilizados e a análise do local de aplicação. Posteriormente, na parte de *firmware*, já estando selecionados os microcontroladores e o microprocessador, foram determinadas todas as rotinas de operação e escolhidas, respectivamente, as linguagens para programá-los e o sistema operacional a ser utilizado no microprocessador. Na parte de *software*, foram selecionadas as ferramentas para a criação de interfaces dentro da camada *front-end*: aplicações *mobile* e *desktop*.

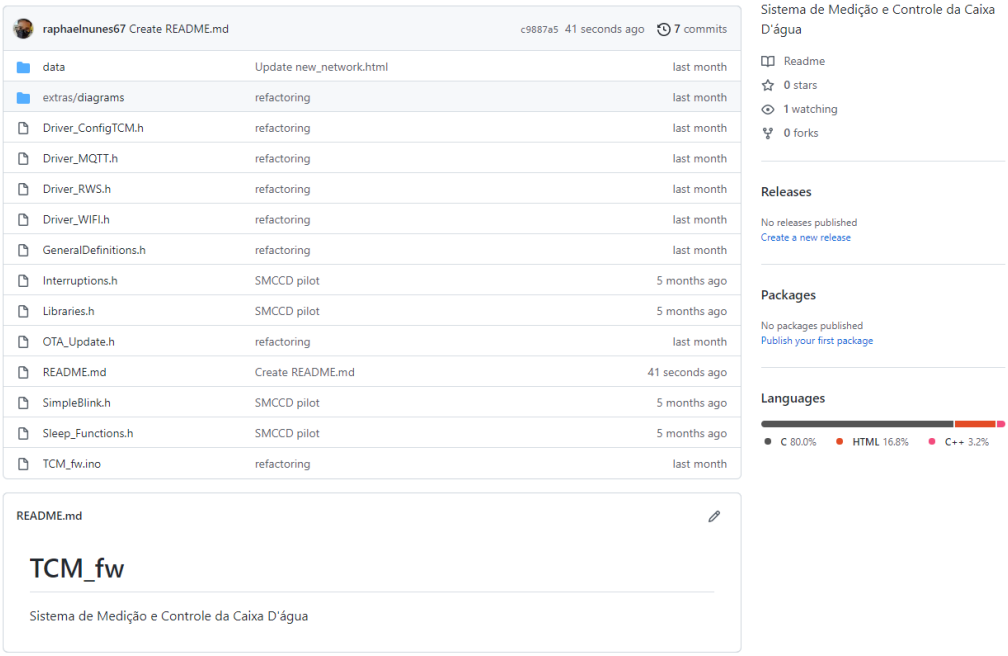
Por fim, elaborou-se a lista de materiais necessários para construção o projeto, tendo objetivo de validação em ambiente real, efetuando testes de longos períodos, averiguando a integridade do sistema, a robustez dos componentes e a identificação de casos não previstos anteriormente, tornando possível a aplicação de melhorias posteriores a entrega deste trabalho.

Figura 7.5 – Visão geral do quadro *Kanban* criado na ferramenta *Trello*



Fonte: Própria.

Figura 7.6 – Visão geral repositório criado no *Github*



Fonte: Própria.

7.3 Levantamento do referencial bibliográfico e capacitação

Essa parte do trabalho conteve-se no levantamento de referências bibliográficas relacionadas com os temas de *IoT*, programação de microcontroladores, criação de aplicações com *frameworks* baseados em *Javascript*. Também buscou-se a capacitação em cursos oferecidos pelas plataformas *Alura*, *Udemy* e *Skylab (Rocketseat)* o aprendizado de conteúdos complementares ao curso de formação em engenheria de controle e automação, como a criação de placas de circuito impresso, treinamentos sobre *Linux* embarcado, implementação do protocolo *MQTT*, criação de *APPs Android* com *React Native* e aplicações *Desktop* com *Electron.js*.

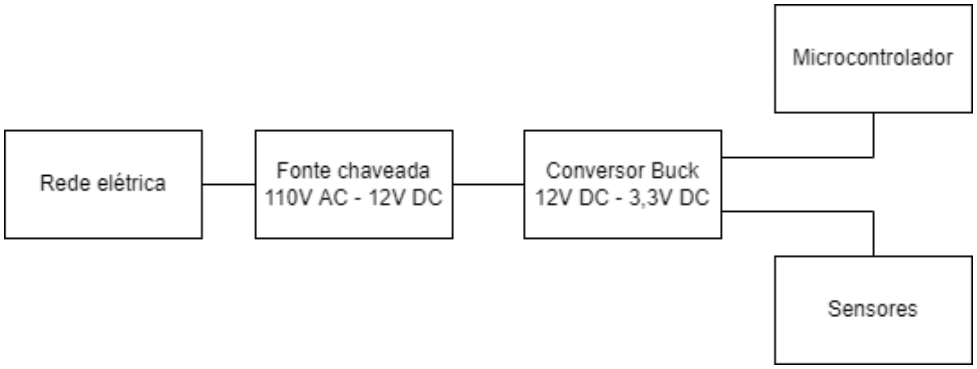
7.4 Desenvolvimento dos elementos de *Hardware*

Nesse momento foram definidos todos os elementos de *hardware* necessários para o desenvolvimento do trabalho: dispositivos eletrônicos, documentos como *Datasheets*, catálogos e informativos de dispositivos elétricos. Realizou-se uma busca no mercado pelos componentes necessários, posteriormente, efetuando compras para a execução de testes.

7.4.1 O circuito de alimentação

Nesta etapa do projeto iniciou-se a criação do circuito de alimentação para os módulos **CCM** e **TCM**. Partindo-se da fonte chaveada definida no Capítulo 6, a qual transforma 110V AC em 12V DC, criou-se um conversor DC-DC do tipo *Buck* para converter a tensão de saída da fonte para 3.3V, ideais para alimentação dos sensores e microcontroladores.

Figura 7.7 – Esquema de ligação do circuito de alimentação

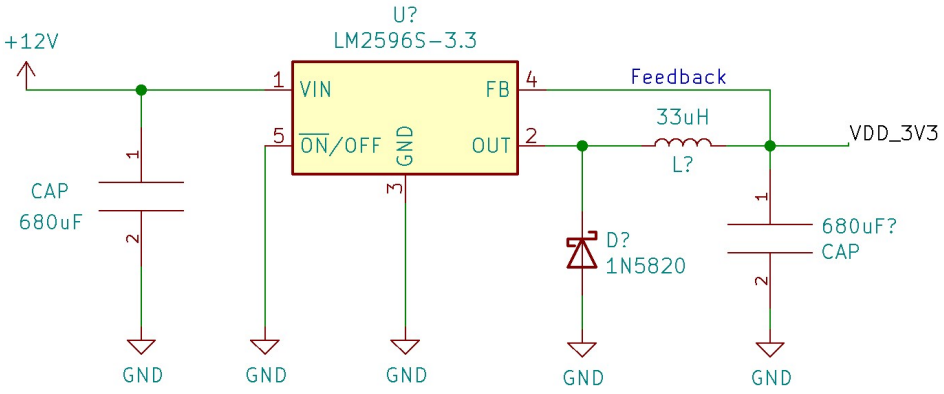


Fonte: Própria.

Após a definição do esquema de medição, procurou-se dimensionar o conversor *Buck*. O dimensionamento deu-se a partir das leituras dos *datasheets* de todos os dispositivos que futuramente poderiam ser utilizados. Outro ponto importante salientar para a escolha do conversor *Buck* foi a disponibilidade no mercado.

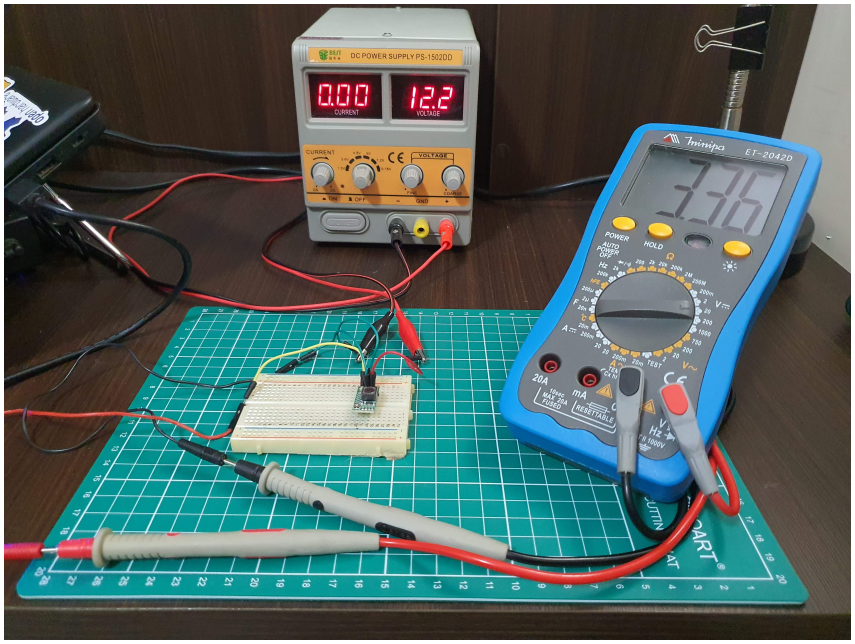
Dentre os fatores citados o conversor selecionado foi o LM2596, o qual podemos encontrar um módulo com sua aplicação típica (Figura 7.8) com certa facilidade no mercado. A Figura 7.9 mostra o teste realizado em bancada.

Figura 7.8 – Aplicação típica do LM2596.



Fonte: Própria.

Figura 7.9 – Teste em bancada do módulo com LM2596.

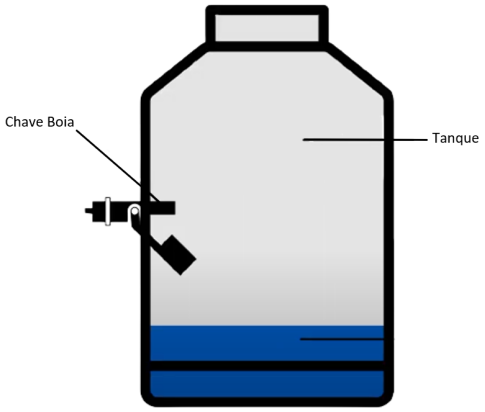


Fonte: Própria.

7.4.2 O circuito de detecção de fechadura da chave

Garantindo a redundância do sistema, no quesito de segurança, planejou-se a aplicação de uma chave tipo boia para ser acoplada a caixa d'água do módulo TCM (Figura 7.10). Essa chave tem como objetivo garantir que não ocorra vazamento durante o processo de enchimento do tanque.

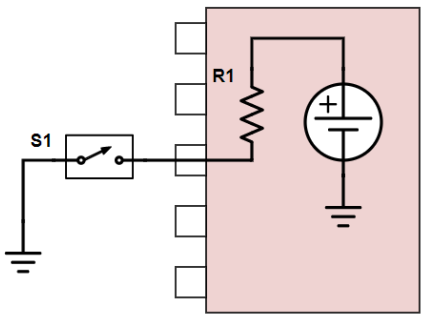
Figura 7.10 – Esquema de aplicação da chave boia



Fonte: Própria.

Para executar a detecção com essa chave de dois estados foi-se necessário aplicar um circuito com resistores de *pullup* (Figura 7.11).

Figura 7.11 – Esquema de ligação elétrica da chave boia.

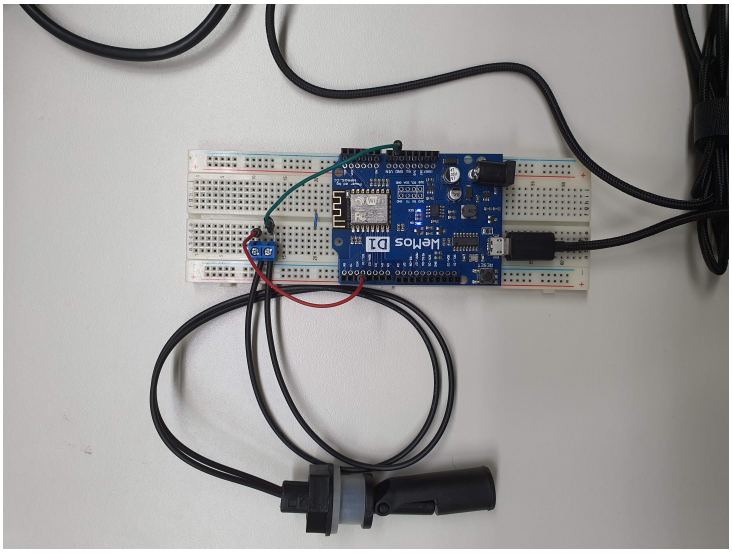


Fonte: Própria.

Ao definir um pino do microcontrolador como do tipo entrada (*INPUT*), e se em certo momento não houver nada conectado à esse pino, é impossível determinar qual o nível lógico será lido. O esquema de pullup tem o intuito de eliminar tal risco garantindo que apenas dois valores bem definidos sejam lidos no terminal: *HIGH* ou *LOW*.

Em alguns microcontroladores, como no ESP8266, existem alguns pinos que podem se programados como do tipo *pullup*, garantindo que não seja necessário montar um circuito externo para isso. A Figura 7.12 mostra a montagem realizada em bancada para o teste da chave boia, conectada ao pino digital 5.

Figura 7.12 – Chave boia acoplada ao ESP-12S através de um pino digital com *INPUT PULLUP*.



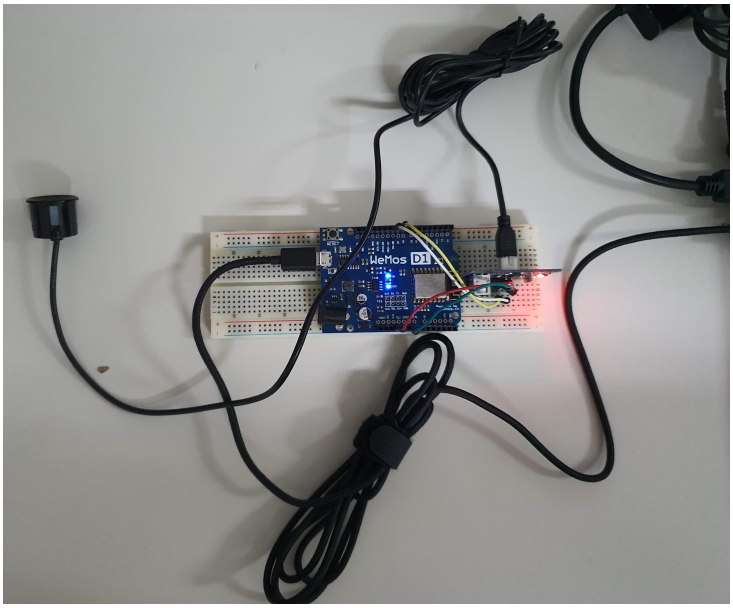
Fonte: Própria.

7.4.3 O circuito de medição de nível

Tendo como base o módulo JSN-SR04T, realizou-se os testes para o envio e recebimento de dados (Figura 7.13). Como descrito na fundamentação teórica, esse módulo possui três tipos de operação, que são selecionadas a partir de um resistor soldado à placa (R27). O modo de operação escolhido para esse projeto foi o terceiro, logo um

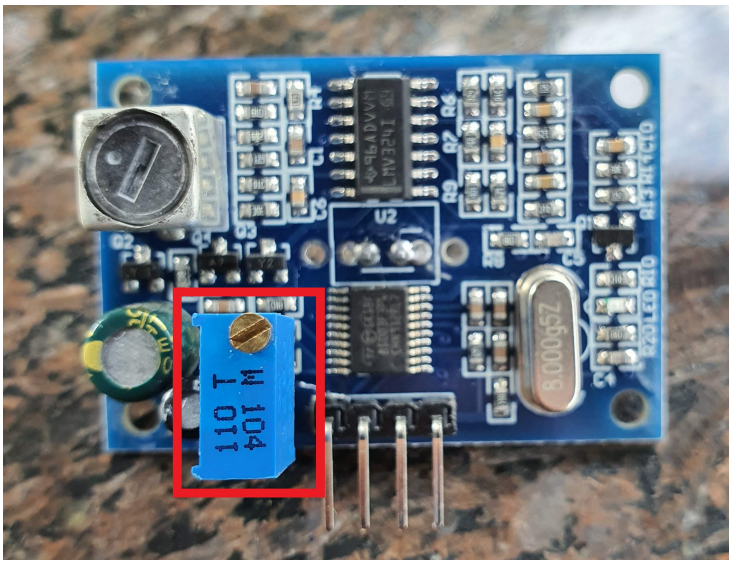
potenciômetro de $120k\Omega$ (ajustado em $47k\Omega$.) foi soldado à placa (Figura 7.14). Esse sensor deve estar acoplado, para executar uma leitura precisa, na parte superior do tanque com o seu transceptor apontando diretamente para o líquido.

Figura 7.13 – Teste de ligação sensor do módulo JSN-SR04T.



Fonte: Própria.

Figura 7.14 – Potenciômetro soldado ao módulo.

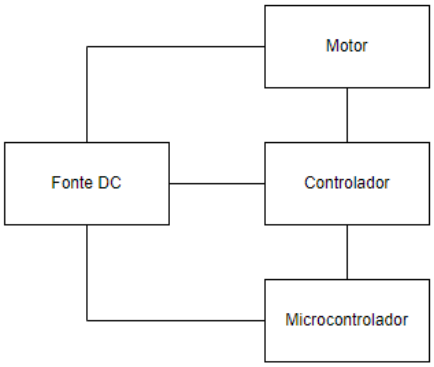


Fonte: Própria.

7.4.4 O circuito de ativação da motobomba

Nesta etapa do projeto deu-se inicio a criação do controlador da motobomba. O diagrama a abaixo mostra um visão geral dos itens que precisaram ser integralizados para a ativação e desativação do motor por meio do microcontrolador.

Figura 7.15 – Diagrama de ativação do motor



Fonte: Própria.

Como visto, a motobomba selecionada opera com 12V em corrente contínua e, por meio de testes feitos, possui um consumo médio de 10A quando está em operação. (Figura 7.16).

Figura 7.16 – Teste da motobomba

Fonte: Própria.

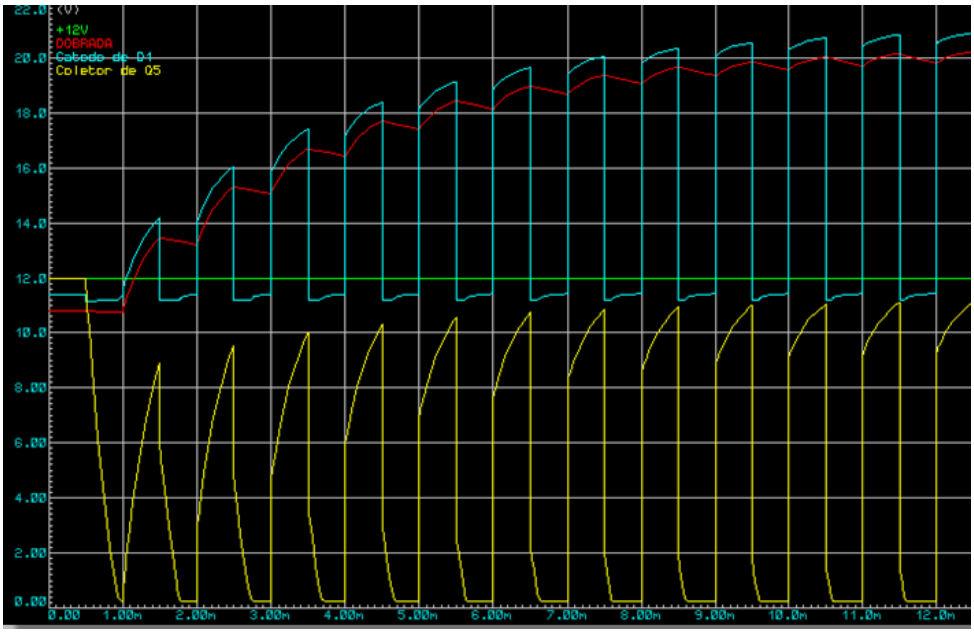
Com os dados obtidos através dos testes, deu-se início a procura de componentes e metodologias para o controle *ON-OFF* de motores de corrente contínua. Como o microcontrolador não possui tesão nem corrente suficientes para ativar o motor, foi-se necessário desenvolver uma ativação indireta transistorizada.

A primeira parte do esquema (Figura 7.19 [B]) conta com o transistor MOSFET IRF1404 de alta potência. Esse transistor, segundo seu *datasheet* (recorte presente no **anexo C**), pode operar com uma corrente máxima de dreno igual a 75A e necessita de uma diferença mínima de 4V entre o V_{GS} e V_{DS} para conduzir a corrente I_{DS}. Com essas informações chegamos a conclusão de que a tensão aplicada ao GATE deveria ser, de no mínimo, 14V.

Partindo-se disso, procurou-se técnicas para elevação de tensão e selecionamos construir um **dobrador de tensão**. Por ser simples e barato para ser desenvolvido, existem inúmeras aplicações que utilizam esse método para ativação de transistores MOSFET. Este circuito é baseado em resitores, capacitores, diodos e necessita de um oscilador conectado na base de transistor para executar ações de chaveamento.

A Figura 7.17 mostra o comportamento da saída do dobrador montado (Figura 7.18). Onde temos a saída de tensão (em vermelho) e a entrada de tensão 12V da fonte DC (em verde). É importante salientar que a tesão de saída atingiu valores em torno de 22V devido as perdas dos componentes, porém valor suficiente para ativar o IRF1404. As ações de chaveamento foram feitas através de *Pulse-Width Modulation - PWM* originados do ESP-12S.

Figura 7.17 – Resultados obtidos da leitura de pontos estratégicos do dobrador de tensão montado



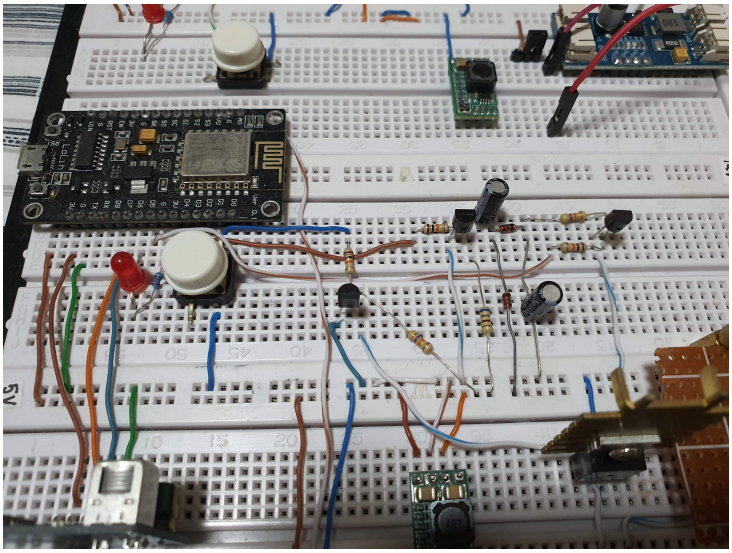
Fonte: Própria

A próxima etapa de implementação do controlador *ON-OFF* foi interligar a saída do dobrador de tensão com o GATE do IRF1404. Para isso utilizamos um circuito com o transistor BC546 (Figura 7.19 [C]), o qual garante que um pino digital do microcontrolador possa ativar ou desativar a motobomba. Outra característica dessa ligação é possibilitar um *soft start*[1], evitando picos de corrente e melhorando a vida útil do motor.

Por fim, analisando a Figura 7.19 [C] é correto afirmar que o motor será ativado quando se aplicar um nível lógico baixo no transistor Q3 e desativado ao se aplicar nível lógico alto.

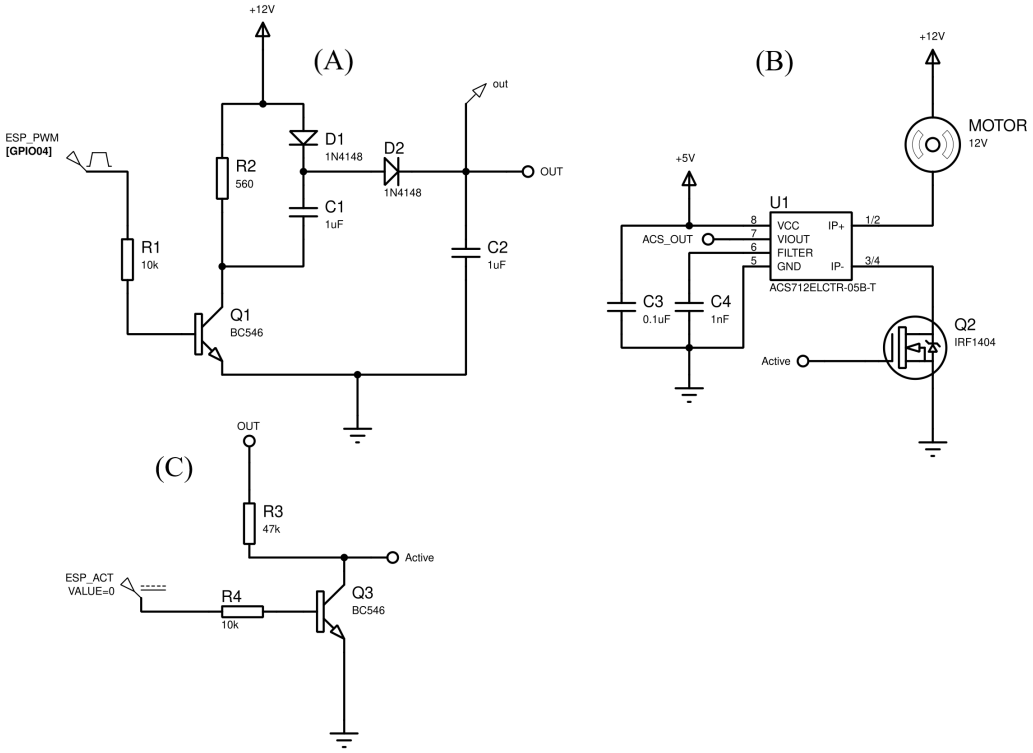
[1]**soft start:** técnica que diminui a corrente de partida, diminuindo os choques mecânicos do motor por meio de uma aceleração constante.

Figura 7.18 – Circuito do dobrador de tensão montado.



Fonte: Própria

Figura 7.19 – Diagrama de ativação com partida lenta



Fonte: Própria

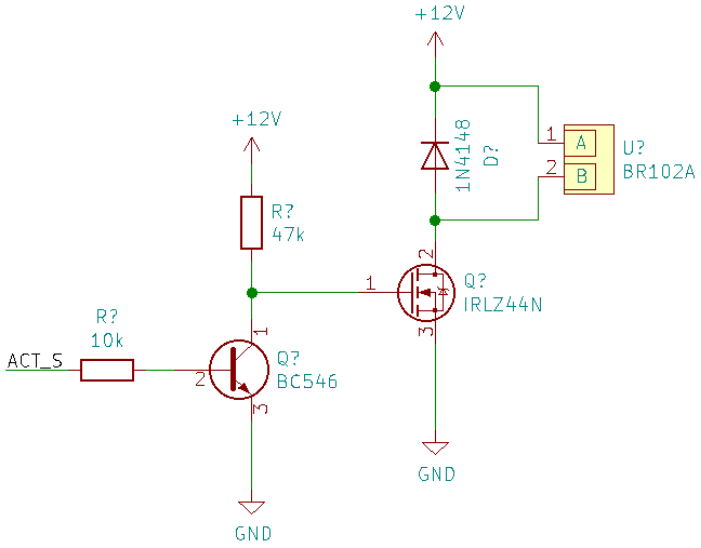
7.4.5 O circuito de ativação da válvula

Assim como o motobomba, a válvula solenoide da Nascimetal selecionada é ativada com tensão de 12V, necessitando também, de acordo com seu catálogo, de uma corrente em torno de $500mA$.

O circuito proposto para a ativação dessa válvula tem como dispositivo principal o transistor MOSFET IRLZ44N. Ele possui a capacidade de operar com corrente de dreno próxima ao valor de 20A, de acordo com seu *datasheet*.

Para efetuar o controle da válvula a partir de um pino digital do microcontrolador, houve a necessidade de adicionar mais um circuito transistorizado, utilizando o BC546 novamente.

Figura 7.20 – Diagrama de ativação da válvula solenoide.



Fonte: Própria

Novamente, observando a Figura 7.20, percebe-se que a ativação da válvula é feita em nível lógico baixo ao aplicar no marcador ACT_S.

7.5 Desenvolvimento dos elementos de *Firmware*

A definição e desenvolvimento dos elementos de *Firmware* se deu após todos os levantamentos de requisitos do trabalho. Esse momento concentrou-se na escolha de tecnologias mais fáceis de implementação (citadas na fundamentação teórica), que tivessem uma gama de documentações e que fossem empregadas em produtos oficiais. Para lidar com o uso dos microprocessadores e microcontroladores, foram feitas pesquisas visando consolidar conceitos aprendidos durante o curso de graduação.

7.5.1 Instalação e configuração do *Broker MQTT*

No primeiro momento desta etapa pesquisou-se os principais serviços de *broker* disponíveis, tendo em vista pontos como documentação e baixo custo. Com isso o serviço selecionado foi o Eclipse Mosquitto™.

A configuração do *broker* leva em consideração quatro informações: **endereço de IP**, **porta**, **usuário** e **senha**.

Estando a Raspberry Pi Zero W rodando o sistema operacional DietPi e possuindo conexão com a *internet*, a instalação do *broker* e dos serviços de *clients* foram realizadas através dos seguintes comandos no terminal:

```
sudo apt-get install mosquitto
```

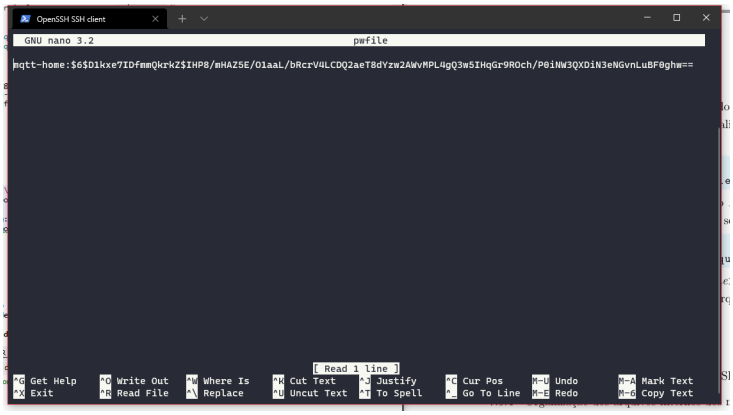
```
sudo apt-get install mosquitto-clients
```

Buscando garantir a segurança de acesso ao *broker* realizou-se os seguintes passos no terminal a fim de configurar um usuário com senha:

```
sudo stop mosquitto
sudo mosquitto_passwd -c /etc/mosquitto/passwd <user_name>
```

O usuário é definido em *<user_name>* e a senha foi solicitada posteriormente. Esses dados são criptografados e salvos no arquivo de configuração presente em: **/etc/mosquitto/mosquitto.conf** (Figura 7.21).

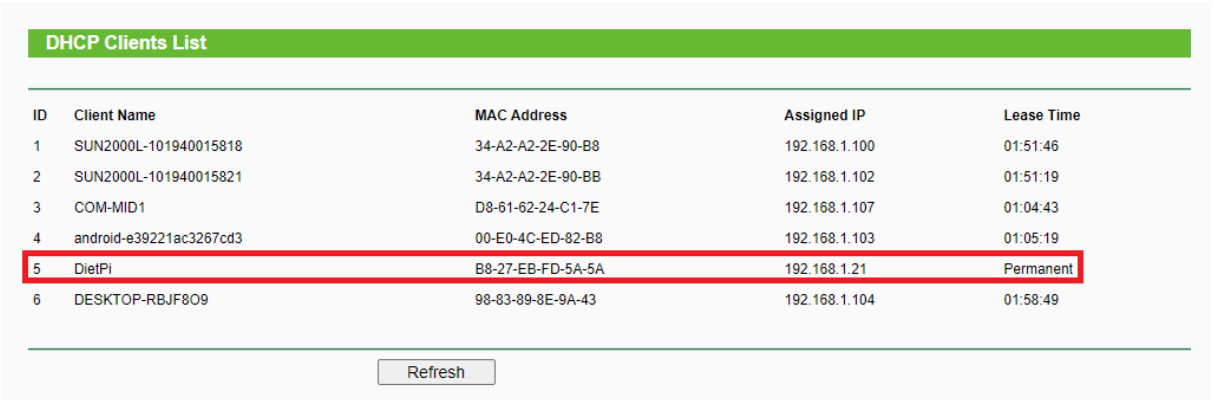
Figura 7.21 – Arquivo de credenciais do Mosquitto gerado



Fonte: Própria

Para que os clientes executem a conexão com o servidor *broker* também foi necessário configurar um IP estático. Essa operação se deu acessando as configurações do roteador local (Figura 7.22).

Figura 7.22 – Em destaque, IP estático configurado no roteador local.



Fonte: Própria

7.5.2 Definição de utilização dos pinos do ESP-12S

Neste estágio do projeto, organizou-se a ligação de cada pino do microcontrolador esp8266, tendo em vista as características citadas no **Anexo A**. A figura abaixo mostra a organização final.

Figura 7.23 – Declarações dos pinos realizadas no código do módulo **CCM**.

```
9   //Define pins
10  const int WIFI_LED_pin = 12; //Led Red
11  const int MQTT_LED_pin = 2; //Led Blue
12  const int INTERRUPT_pin = 0;
13  const int MOTOR_pin = 14;
14  const int SOLENOID_pin = 5;
15  const int CURRENT_SENSOR_pin = 17; // A0
16  const int WL_ECHO_pin = 13;
17  const int WL_TRIG_pin = 15;
18  const int GEN_pin = 4;
```

Fonte: Própria

Figura 7.24 – Declarações dos pinos realizadas no código do módulo **TCM**.

```
9   //Define pins name
10  const int WIFI_LED_pin = 12; // Red
11  const int MQTT_LED_pin = 2; //Blue
12  //const int AP_LED_pin = 2; // Yellow
13  const int INTERRUPT_pin = 0;
14  const int SOLENOID_pin = 4;
15  const int FLOATSWITCH_pin = 5;
16  const int WL_ECHO_pin = 15;
17  const int WL_TRIG_pin = 13;
```

Fonte: Própria

De acordo com as figuras, pontuamos:

- **WIFI_LED_pin**: define o pino de conexão *WI-FI*;
- **MQTT_LED_pin**: define o pino de indicação de conexão MQTT;
- **INTERRUPT_pin**: define o pino de interrupção para execução de *Reset*;
- **MOTOR_pin**: define o pino de ativação e desativação da motobomba;
- **GATE_pin**: define o pino de ativação e desativação da válvula solenoide;
- **FLOATSWITCH_pin**: define o pino da chave do tipo boia;
- **CURRENT_sensor_pin**: define o pino de leitura do sensor de corrente (pino analógico);
- **WL_ECHO_pin**: define um pino de comunicação com o módulo ultrasônico JSN-SR04T;
- **WL_TRIG_pin**: define um pino de comunicação com o módulo ultrasônico JSN-SR04T;

7.5.3 Organização dos arquivos internos dos microcontroladores

Utilizando uma biblioteca chamada *SPIFFS*, nesta etapa do desenvolvimento do *firmware* dos módulos, foi implementado um sistema de arquivos na memória *flash* dos microcontroladores ESP-12S.

Esse sistema de arquivos, gravados na memória não-volátil, tem como objetivo armazenar dados pertinentes como: a versão do *firmware*, os dados para conexão WI-FI e MQTT, além de armazenar páginas *HTML* para o processo de cadastro, o qual será descrito no quesito 7.5.6.

Arquivo	Função
system_info.json	armazena a versão do firmware, versão do sistema de arquivos, data do ultimo update e o link para o repositório de novos binários
register_config.json	armazena os dados para conexão WI-FI (*ssid* e senha) e os dados para conexão ao *broker* MQTT (IP, porta, usuário, senha, tópico de inscrição e o tópico de publicação)
new_network.html	página para cadastro de do dispositivo na rede WI-FI e ao servidor MQTT
confirm.html	página de confirmação para recebimento dos dados após o cadastro

Tabela 7.2 – Descrição dos arquivos salvos na memória interna do microcontrolador.

7.5.4 Comunicação com o sensor de nível

Como visto anteriormente módulo JSN-SR04T possui três modos de operação. Para este trabalho selecionou-se o modo **3** por utilizar o controlador da placa e por permitir um baixo consumo. A função evidenciada na Figura 7.25 é chamada sempre que se deseja realizar uma leitura de distância. Ela envia o comando 0x55 via serial e aguarda a resposta do módulo. Quando a resposta chega, ela é segmentada e armazenada na variáveis: *startByte*, *h_data*, *l_data* e *sum*. A validação é feita através da comparação:

```
if ((startByte + h_data + l_data) != sum)
```

Se o valor a comparação for igual a função retorna o valor lido em milímetros, se não for retorna o valor zero.

Figura 7.25 – Função para realizar a leitura de distância a partir do sensor ultrassônico.

```
32  // Get Water Level(Sensor SR04T)
33  String GetWaterLevel() {
34    jsnSerial.write(0x55);
35    delay(50);
36    while True{
37      if(jsnSerial.available()){
38        unsigned int distance;
39        byte startByte, h_data, l_data, sum = 0;
40        byte buf[3];
41        startByte = (byte)jsnSerial.read();
42
43        if(startByte == 0xFF){
44          jsnSerial.readBytes(buf, 3);
45          h_data = buf[0];
46          l_data = buf[1];
47          sum = buf[2];
48
49          distance = (h_data<<8) + l_data;
50          if (((startByte + h_data + l_data)&0xFF) != sum){
51            return String(0);
52          }
53          else{
54          return String (distance);
55          }
56        }
57      }
58    }
59  }
60  // END - Get Water Level
```

Fonte: Própria

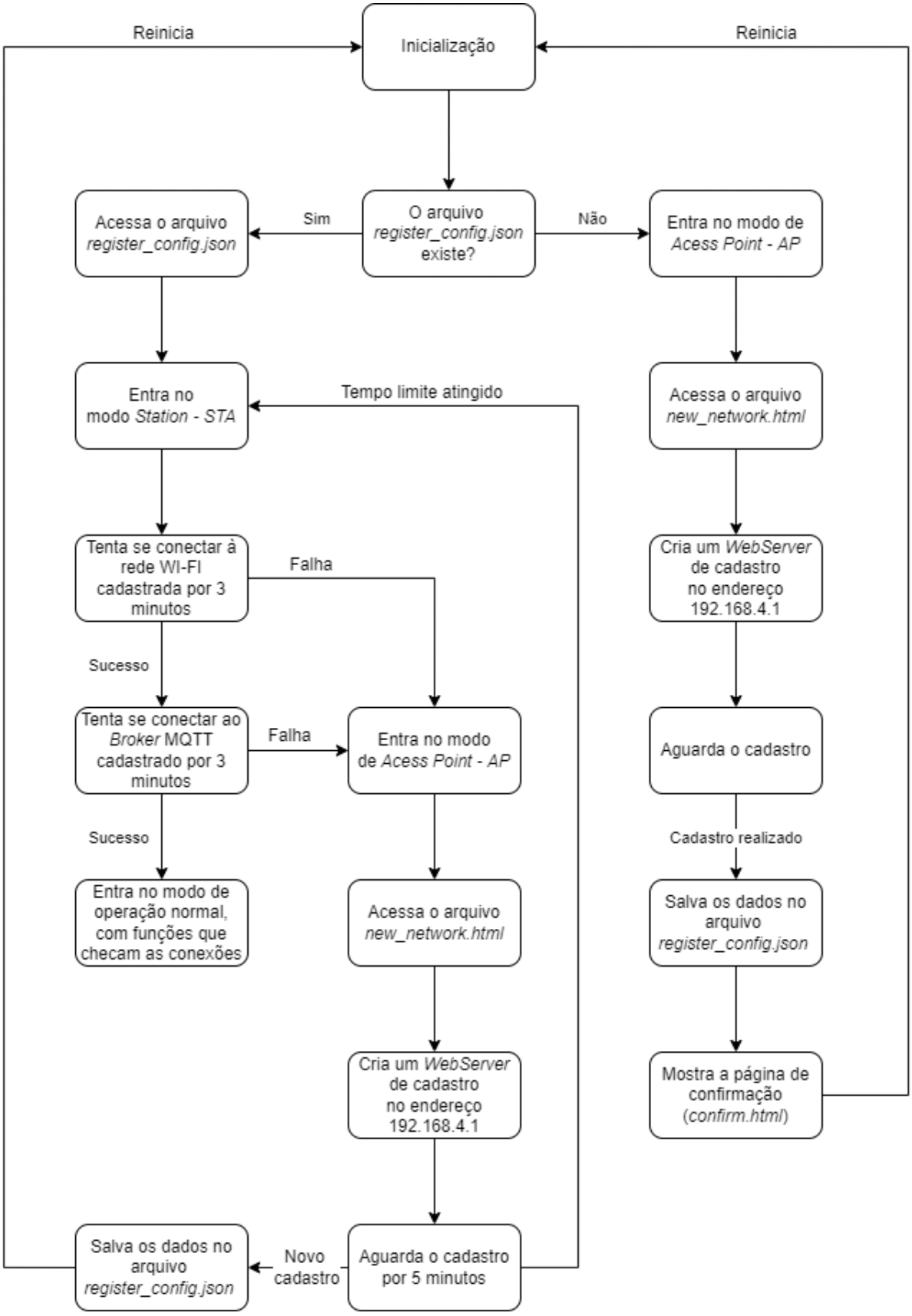
7.5.5 Implementação do *Driver WI-FI*

Esta seção concentra-se em explicar o conjunto de funções criadas, organizadas no arquivo de cabeçalho *Driver_WIFI.h*, as quais lidam com as rotinas de conexão e desconexão assim como as operações de cadastro do dispositivo. Um módulo, seja ele um **TCM** ou **CCM**, na sua primeira utilização será necessário configurar: a rede *WI-FI* para se conectar, o servidor *broker* e os tópicos de inscrição e de publicação. Esse cadastro é realizado a partir de uma página *HTML* (Figura 7.27) gerada pelo microcontrolador, que nos estágios iniciais se comporta como um ponto de acesso (*Access Point)*. É importante notar que se o módulo perder conexão com o *WI-FI* ou com o servidor $MQTT$ ele irá entrar no modo de alternância: 3 minutos tentando uma reconexão e 5 minutos com um ponto de acesso disponível para uma nova configuração.

O diagrama da Figura 7.26 mostra o comportamento completo do *firmware* após o *boot* [2] do microcontrolador.

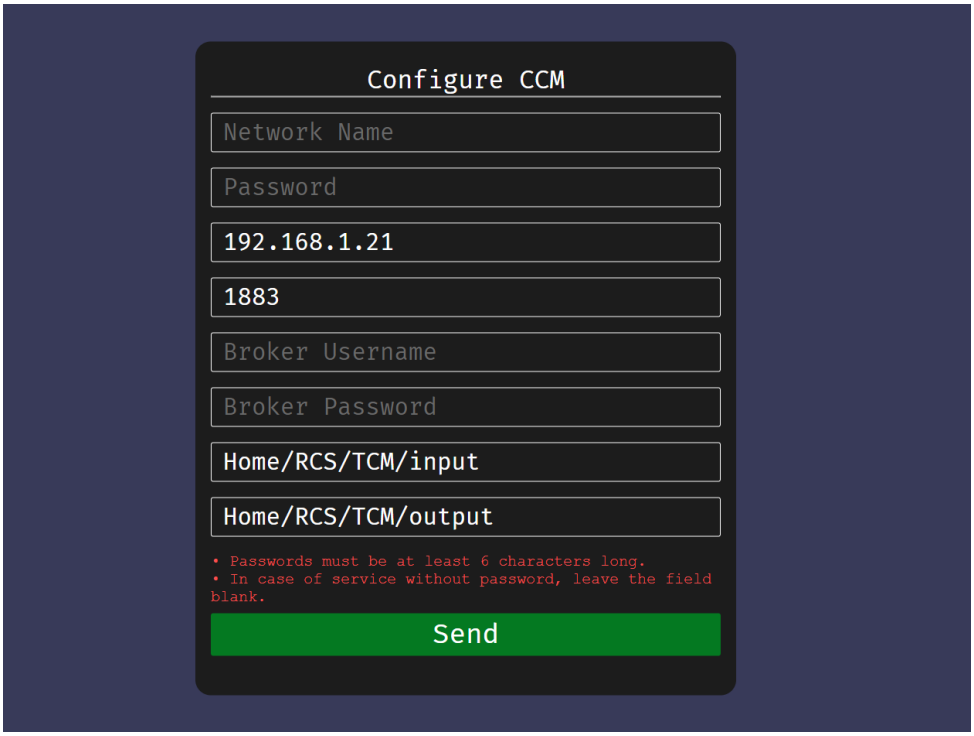
[2] ***boot***: termo utilizado para fazer referência ao processo de inicialização de um dispositivo microcontrolado ou microprocessado.

Figura 7.26 – Diagrama de operação do microcontrolador.



Fonte: Própria

Figura 7.27 – Página *HTML* gerada para cadastro do dispositivo.



Fonte: Própria

7.5.6 Implementação do *Driver MQTT*

Estando um módulo devidamente cadastrado, em modo normal de operação, ele utiliza as funções de *MQTT*. A biblioteca que permite as ações de conexão, publicação e inscrição com o procolo *MQTT* se chama *PubSubClient*. Com a utilização desse biblioteca foi-se possível configurar o $\boldsymbol{QoS} = 2$, assegurando uma troca de mensagens de forma mais confiável.

Nesta etapa da produção de código, foi criado um *callback* o qual executa ações de acordo as mensagens que chegam no tópico em o *client* **TCM** ou **CMM** está inscrito. Esse callback pode é apresentado na Figura 7.28.

Figura 7.28 – Função de *Callback*.

```
42  // Callback MQTT payloads
43  void callback(char* topic, byte* payload, unsigned int length) {
44  #ifdef DEBUG
45    Serial.print("Message arrived [");
46    Serial.print(topic);
47    Serial.print("] ");
48  #endif
49    payload[length] = '\0';
50    String cb_message = String((char*)payload);
51  #ifdef DEBUG
52    Serial.println(cb_message);
53  #endif
54
55
56    if (cb_message == "OPEN"){CommandSolenoid(false);}
57
58    else if (cb_message == "CLOSE"){CommandSolenoid(true);}
59
60    else if (cb_message == "UPDATE"){SendValues();}
61
62    else if (cb_message == "GS"){ESP_Sleep();}
63
64    else if (cb_message == "OTA"){OTA_https_update();}
65
66    else if (cb_message == "SI"){Send_system_info();}
67
68    else if(cb_message.indexOf("http") != -1){UpdateVC_link(cb_message);}
69  }
70  //END - Callback MQTT payloads
```

Fonte: Própria

Ao receber uma mensagem no tópico em que está inscrito, o programa valida a mensagem que deve ser e executa as ações de acordo com as condições.

Menssagem	Ação
ON	Liga a motobomba
OFF	Desliga a motobomba
OPEN	Abre a válvula solenoide
CLOSE	Fecha a válvula solenoide
UPDATE	Retorna um arquivo json com os dados sesores
GS	Coloca o processador em modo de DeepSleep
OTA	Verifica se há uma atualização de firmware disponível
SI	Retorna o um arquivo json com os dados do sistema
HTTP + LINK	Cadastra um novo link para busca de atualizações

Tabela 7.3 – Comandos válidos para os módulos TCM e CCM

7.5.7 Implementação do *Factory Reset*

Como visto nas seções anteriores, as ações do microcontrolador são tomadas a partir do arquivo de configuração *register_config.json*. A rotina de *Factory Reset* [3] implementada tem como objetivo levar o microcontrolador ao estado inicial presente na figura. Para que isso aconteça é necessário apenas apagar o arquivo *register_config.json* e reiniciar dispositivo.

Tomou-se como base as implementações de *Factory Reset* de roteadores, as quais são executadas a partir de um botão físico. O código abaixo mostra como ocorre a operação de restauração, utilizando um botão interligado à um pino de interrupção.

[3] **Factory Reset**: procedimento que faz um determinado dispositivo eletrônico retornar ao seu estado original de fábrica.

Figura 7.29 – Função de interrupção.

```
// Interrupt Function [Factory Reset button]
void ICACHE_RAM_ATTR FactoryReset() {
  if (SPIFFS.exists("/register_config.json")) {
    unsigned long PressTime = millis();
    while (digitalRead(INTERRUPT_pin) == LOW) {
      if (millis() - PressTime >= 3000) {
        SPIFFS.remove("/register_config.json");
        ESP.restart();
      }
    }
  }
}
// END - Interrupt Function

// Set Interrupt Function
void SetInterrupt() {
  attachInterrupt(digitalPinToInterrupt(INTERRUPT_pin), FactoryReset, CHANGE);
}
// END - Set Interrupt Function

```

Fonte: Própria

Uma interrupção é um sinal enviado por um dispositivo de *hardware* (compreendido internamente no *chip* do microcontrolador) que temporariamente interrompe a tarefa que a *CPU* está executando no momento, para executar uma ação pré-estabelecida. Logo após, o programa retoma seu processamento do ponto onde havia parado. No caso de interrupções externas, o esp8266 aceita 5 modos distintos:

- CHANGE – Interrupção é disparada quando o pino muda de estado;
- FALLING – Interrupção é disparada quando o pino vai do estado HIGH para LOW;
- HIGH – Interrupção é disparada quando o pino está no nível HIGH;
- LOW – Interrupção é disparada quando o pino está no nível LOW;
- RISING – Interrupção é disparada quando o pino vai do estado LOW para HIGH.

Na Figura 7.29 a função *SetInterrupt()* define qual função deve ser executada durante a interrupção e qual o modo de disparo. A função *ICACHE_RAM_ATTR FactoreReset()* remove o arquivo *register_config.json* e reinicia o microcontrolador se o botão for pressionado por mais de 3 segundos.

7.5.8 Implementação da funcionalidade de *OTA Upgrade*

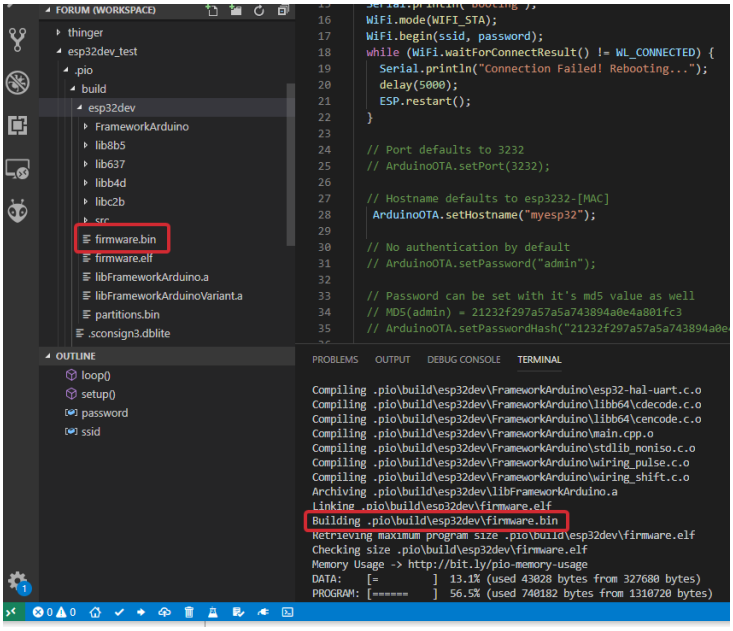
Nesse estágio do projeto buscou-se integrar ao código a funcionalidade de atualização via *Over-The-Air* - OTA. Esse tipo de ação é feita de forma remota, ou seja, para modificar o *firmware* é necessário apenas uma conexão sem fio e elimina-se a utilização de comunicação física (via cabeamento). A eliminação do contato físico também garante que o módulo tenha seu custo de produção reduzido.

O ESP8266, sendo programado com *framework* Arduino, possui duas bibliotecas que lidam com atualizaçoes via OTA.

A biblioteca utilizada neste projeto foi a *ESP8266httpUpdate*. Ela permite que o microcontrolador acesse determinado repositório, faça o *download* do binário e aplique a atualização automaticamente.

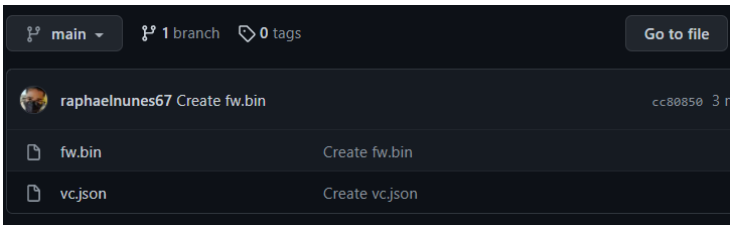
Ao compilar o código, o PlatformIO já gera o arquivo binário em um de seus diretórios padrão (Figura 7.30). Ao gerar esse arquivo, criamos um repositório aberto no *Github* e armazenamos o binário gerado (Figura 7.31).

Figura 7.30 – Binário gerado ao realizar o *build* do código.



Fonte: Própria

Figura 7.31 – Repositório criado.



Fonte: Própria

Na rotina MQTT do microcontrolador, como visto na Tabela 7.3, existem duas mensagens que tratam da funcionaliadade de OTA *Upgrade*. Se a mensagem recebida no tópico for simplemente ”OTA”, o microcontrolador acessa o endereço *url* presente em sua memória. Esse *link* contém um arquivo *.json* com dois parâmetros: a versão do binário presente no repositório e o *link* para o mesmo. Utilizamos essa abordagem de acesso indireto para que não seja necessário atualizar o *link* armazenado na memória interna do ESP frequentemente mas sim o que está presente no repositório.

Se a mensagem recebida pelo microcontrolador, no tópico programado, for um *link*, o mesmo já executa a ação de salvá-la no arquivo interno (*system_info.json*).

7.6 Desenvolvimento dos elementos de *Software*

Para o desenvolvimento dos elementos de *Software* buscou-se a participação de cursos básicos e avançados sobre aplicações *front-end*, as quais servem de conteúdo complementar aos temas abordados na graduação em engenharia de controle e automação.

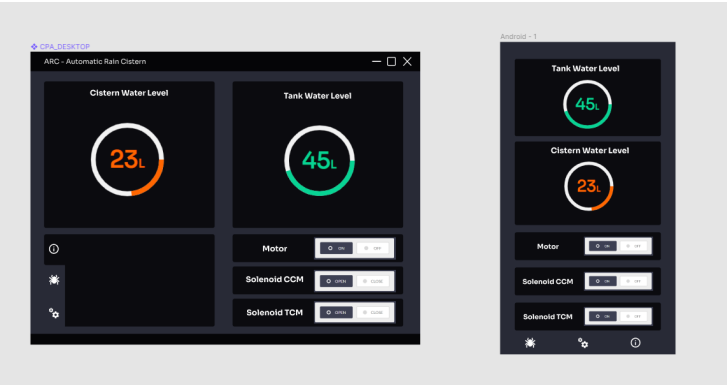
Os cursos adquiridos trouxeram uma série de conhecimentos para uma maior interação entre desenvolvedor e usuário, possibilitando a criação de interfaces intuitivas e

agregando novas informações e visões a outros temas, como na programação de sistemas embarcados.

7.7 Desenvolvimento da aplicações *Desktop* e *Mobile*

O desenvolvimento das aplicações *desktop* e *mobile* tiveram início com a formulação da tela utilizando o *Figma*. Primeiramente selecionamos a paleta de cores para garantir um *software* mais atraente visualmente e a partir disso alguns esboços foram criados. Uma das preocupações, foi garantir que os mostradores e botões fossem bem compreendidos e de fácil acesso, com isso, as telas (Figura 8.1) apresentaram-se convenientes.

Figura 7.32 – Definição do *layout* das telas criado no Figma.



Fonte: Própria

As aplicações rodam em uma única tela com quatro seções, em formato de *dashboard* [4]. Nas seções superiores mostra os valores de volume da cisterna e da caixa d'água auxiliar em litros. Nas seções inferiores encontram-se os botões para ativação e desativação da motobomba, abertura e fechamento do solenoide A (presente no módulo **CCM**) e abertura e fechamento do solenoide B (presente no módulo **TCM**)

Com o *layout's* definidos iniciou-se o processo de criação das aplicações *desktop* e *mobile* como base as tecnologias *Electron*, *React* e *React Native*.

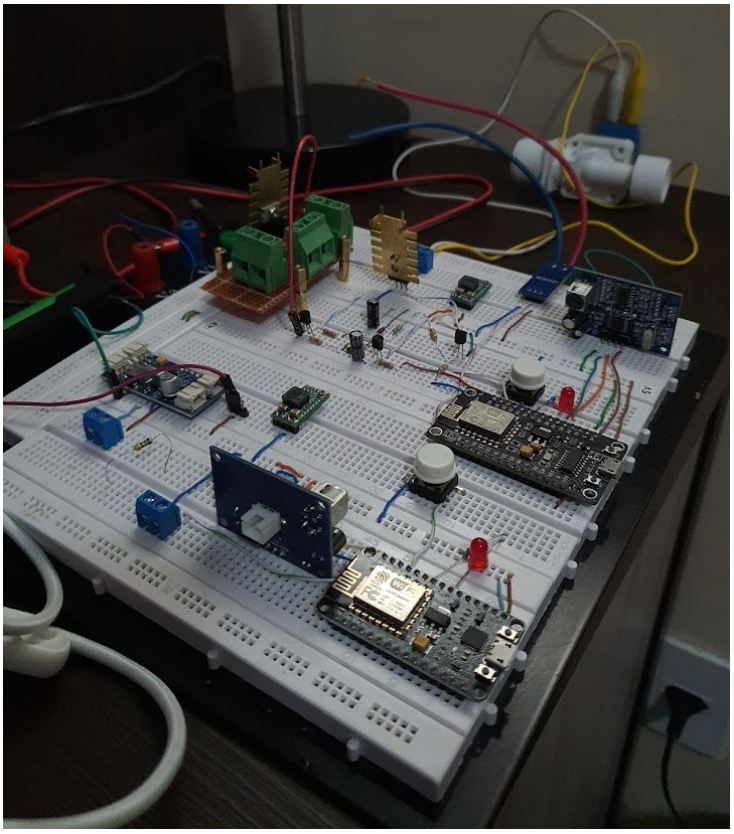
[4] **dashboard**: painel de informação e/ou controle de determinado processo.

8 RESULTADOS

Após a formulação e teste de cada item dos sistemas deu-se início a junção de todas as funcionalidades arquitetadas. O conteúdo deste capítulo visa apresentar os resultados obtidos neste trabalho em cada área de desenvolvimento.

Primeiramente como resultado da parte de *hardware* podemos apresentar a formulação dos circuitos dos módulos **CCM** e **TCM** apresentados nas figuras abaixo. Os dois módulos foram montados em *protoboard* buscando validar, em conjunto, todas funcionalidades as seções descritas na metodologia.

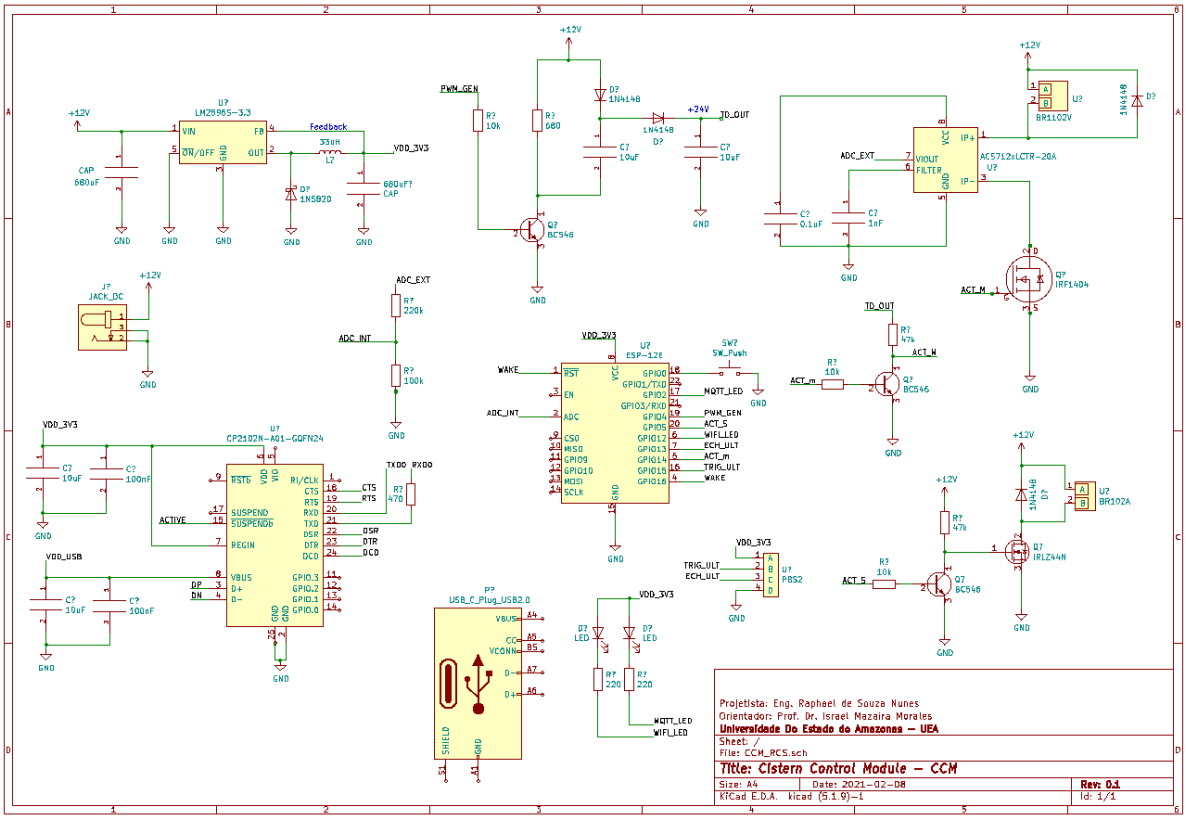
Figura 8.1 – Protótipo montado.



Fonte: Própria

Também foram desenvolvidos os esquemáticos dos dois módulos (Figura 8.2), levando em consideração todas as ligações: circuitos de alimentação, sensoriamento e comunicação. Esse resultado garante futuras implementações de *layout's 3D* e confecção das placas de circuito impresso.

Figura 8.2 – Esquemático do módulo CCM implementado no Kicad.



Fonte: Própria

No quesito de *firmware*, alcançou-se todas as funcionalidades propostas, gerando um código estável e escalável para projetos reais. Os códigos associados podem ser acessados no *Github*. O código principal, Figura 8.3, conta com diversos arquivos de cabeçalho que modularizam as funcionalidades:

1. *Libraries.h*: ficam armazenadas as bibliotecas de terceiros que foram utilizadas;
2. *GeneralDefinitions.h*: são definidos os pinos e são iniciadas as variáveis globais;
3. *Interruptions.h*: é implementado a funcionalidade de *Factory Reset* por meio de interrupção externa (botão);
4. *Sleep_Functions.h* possui funções de modo de *DeepSleep*, porém este modo não é utilizado;
5. *SimpleBlink.h*: define o compotamento dos leds;
6. *Driver_ConfigTCM.h* ou *Driver_ConfigCCM.h* define e manipula os arquivos de configuração do microcontrolador;
7. *Driver_RWS.h*: é utilizado para leitura e escrita no sistema de arquivos interno;
8. *Driver_WIFI.h*: define as funções de conexão e modos de operação do *chip* de WI-FI;
9. *Driver_MQTT.h*: é utilizado para configurar a conexão com o *broker* assim como definir os *callbacks*;
10. *OTA_upgrade.h*: utiliza os métodos para atualização via *internet*;

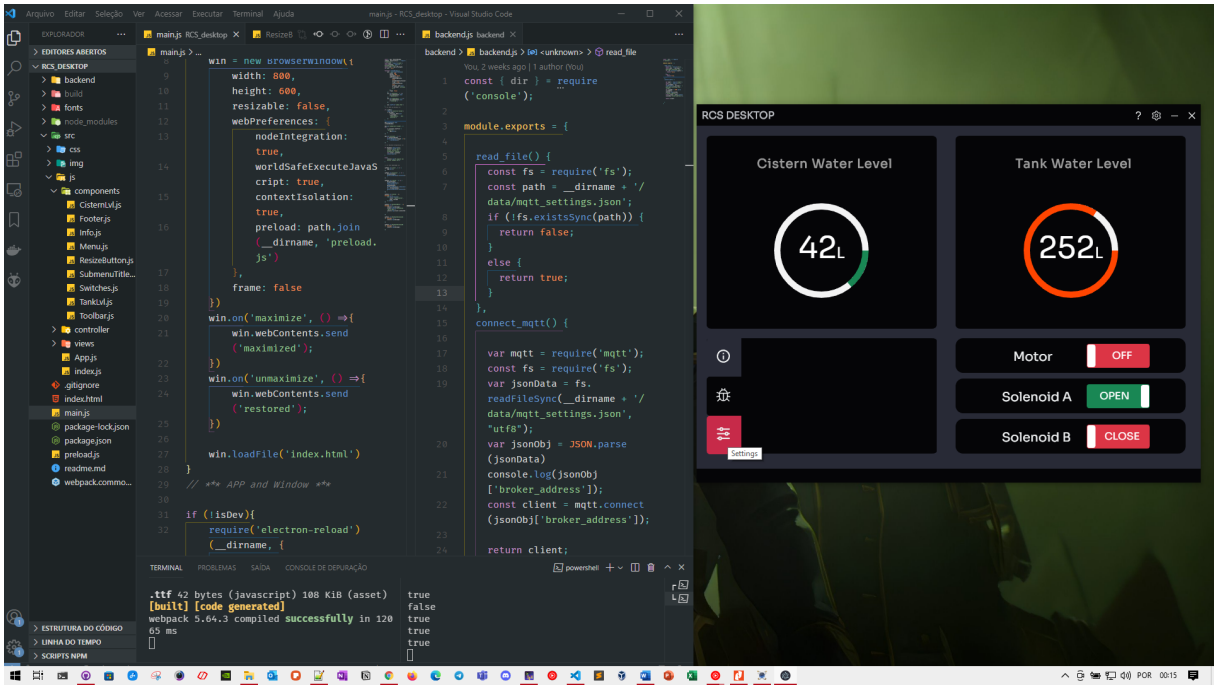
Figura 8.3 – Versão final do código do *firmware*

```
/*
 ***************************************
                TCM_fw.ino
  Script para ESP8266MOD (Caixa D'água)
  Autor: Raphael Nunes
  E-mail: raphaelnunes67@gmail.com
 ***************************************
*/
#define DEBUG
//Includes
#include <Arduino.h>
#include "Libraries.h"
#include "GeneralDefinitions.h"
#include "Interruptions.h"
#include "Sleep_Functions.h"
#include "SimpleBlink.h"
#include "Driver_ConfigTCM.h"
#include "Driver_RWS.h"
#include "Driver_WIFI.h"
#include "OTA_Update.h"
#include "Driver_MQTT.h"


void setup() {
#ifdef DEBUG
  Serial.begin(115200); // Serial debug
#endif
  SPIFFS.begin();
  SetPins();  // Defines the pins and their initial values
#ifdef DEBUG
  ShowSystemVersion();
#endif
  SetInterrupt(); //Defines interrupt routines (Factory reset)
  VerifyConfigJSON(); // Checks whether registration data is saved in the file system
  initWIFI(); //Starts WIFI
  OTA_arduino_update(); //Enables updating via Arduino IDE
  initMQTT();//Starts MQTT comunication
}

void loop() {
  ArduinoOTA.handle(); // Keeps OTA update via arduino IDE always available
  client.loop(); // MQTT loop (callbacks)
  reconnectWIFI(); //Always check the WI FI connection and with the MQTT server
  reconnectMQTT();
}
```

Fonte: Própria

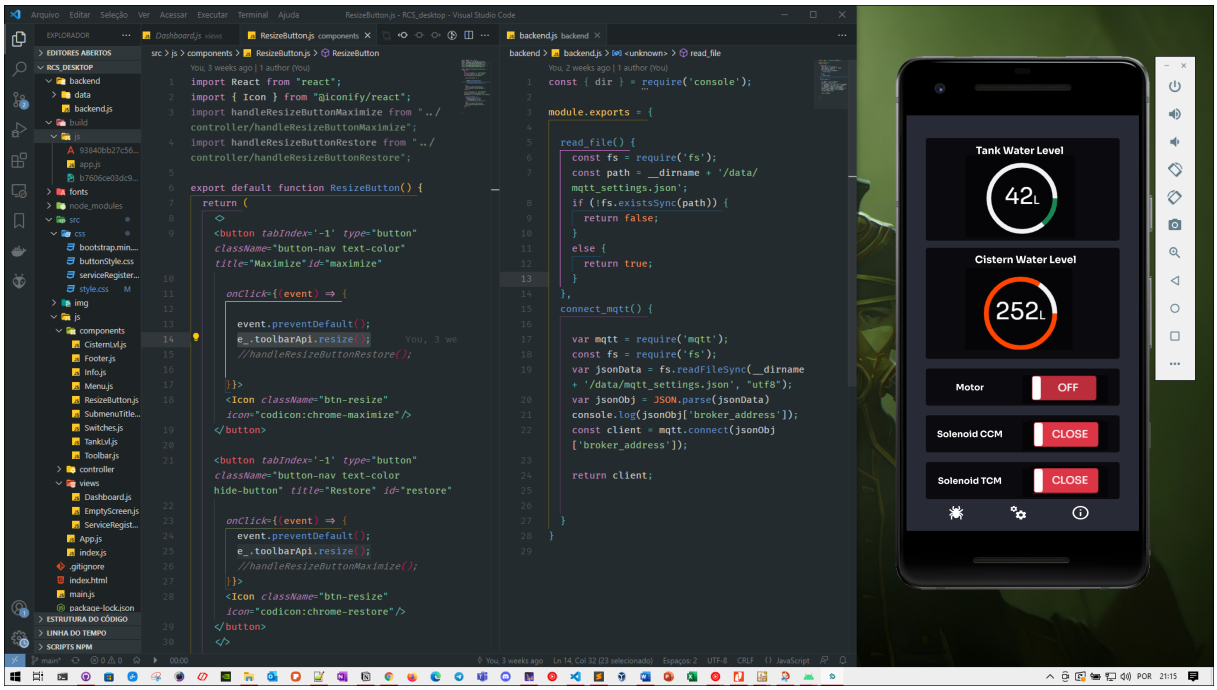
Por fim, também obtivemos como resultado a elaboração de telas utilizando as tecnologias propostas, atingindo o objetivo de relacionar o *software* com as camadas de *firmware* e *hardware*.

Figura 8.4 – Implementação da aplicação *desktop*.



Fonte: Própria

Figura 8.5 – Implementação da aplicação *mobile*.



Fonte: Própria

9 CONCLUSÃO

Este trabalho de conclusão de curso teve como principal objetivo desenvolver um projeto base para aplicação de um sistema automatizado para coleta, armazenamento e distribuição de água da chuva tendo como foco atuar sobre os processos convencionais de uma cisterna. Partindo da estratégia de reunir diversas ferramentas, metodologias e tecnologias estudadas durante o curso de engenharia de controle e automação, houve a possibilidade de que fossem estabelecidos conceitos a serem aplicados em um projeto real.

Para as partes físicas do projeto (*hardware*) considerou-se uma cisterna convencional (não automatizada). A partir de uma esquematização foi possível estabelecer os pontos principais para se aplicar uma certa automação, propondo os módulos **CCM**, atrelado à cisterna e **TCM**, atrelado ao reservatório auxiliar.

Durante o desenvolvimento dos módulos, diversos testes de conceito foram aplicados. Em ambos os módulos foram propostas implementações com válvula solenoides, para alternar o fluxo de água, e medição de nível por meio de sensores ultrassônicos. Individualmente, no **CCM** foi determinada a utilização de uma motobomba *DC* com o planejamento de todo seu circuito de controle e no **TCM** estruturou-se um sistema de segurança utilizando chave do tipo boia (também conhecida como chave de flutuação).

O projeto também englobou elementos de *software* buscando integrações com os ambientes *desktop* e *mobile*. Os *frameworks Electron* e *React* ajudaram a criar moldes de aplicações mais interessantes para um usuário final, possibilitando o controle dos componentes do sistema de forma mais intuitiva.

Com isso, tomando como base os conceitos de Internet das Coisas e utilizando as ferramentas atuais para desenvolvimento de *firmware* e *software* foi-se possível criar um projeto base para automatizar as operações de uma cisterna pluvial.

9.1 Trabalhos Futuros

Neste item destaca-se alguns pontos que podem ser implementados ou melhorados baseados no tema abordado:

- Criar uma rotina para informar as dimensões dos tanques os quais se desejar medir o volume;
- Adicionar a funcionalidade de controle automático com base nas variáveis do sistema (como, por exemplo, o volume do tanque) e de acordo um horário estabelecido;
- Criar uma forma de autenticação ao repositório de novos binários para atualização de *firmware*;
- Confeccionar a s placas de circuito impresso (*PCB's*) e aplica-las à um sistema real;
- Implementar ou definir as peças necessárias para fixar os sensores, conectores e placas dos módulos;
- Gerar os arquivos de distribuição da aplicação *desktop*;
- Gerar os arquivos de distribuição da aplicação *mobile*;

REFERÊNCIAS

BIZMEET. **Gartner aponta tendências estratégicas para 2017 na área de TI**. 2017. Disponível em: <http://news.bizmeet.com.br/>. Acesso em: 17 set. 2020. 12

CHROMIUM.ORG. **Chrome on Windows performance improvements and the journey of Native Window Occlusion**. 2021. Disponível em: <https://blog.chromium.org/>. Acesso em: 17 set. 2021. 39

CITISYSTEMS. **Como funciona a válvula solenoide e quais seus tipos?** 2017. Disponível em: <https://www.citisystems.com.br/valvula-solenoide/>. Acesso em: 30 nov. 2021. 32

DIAS, R. R. de F. **internet das Coisas Sem Mistérios: Uma nova inteligência para os negócios**. [S.l.]: NETPRESS BOOKS, 2016. 22

ELECTRONICSTUTORIALS. **Hall Effect Sensor**. 2021. Disponível em: <https://www.electronics-tutorials.ws/electromagnetism/hall-effect.html>. Acesso em: 17 set. 2021. 34

ELECTRONJS.ORG. **Quick Start**. 2021. Disponível em: <https://www.electronjs.org/docs/latest/tutorial/quick-start>. Acesso em: 17 set. 2021. 39

EMBARCADOS. **Apresentando o módulo ESP8266**. 2015. Disponível em: <https://www.embarcados.com.br/modulo-esp8266/>. Acesso em: 19 nov. 2021. 74

ESP8266.NET. **About ESP8266**. 2021. Disponível em: <http://esp8266.net/>. Acesso em: 02 set. 2021. 27

FABIOBRANDAO. **O que é MQTT**. 2019. Disponível em: <https://fabiobrandao.net.br/blog/tecnologia/o-que-e-mqtt/>. Acesso em: 02 set. 2021. 25

FERNANDO K. **NodeMCU ESP8266: Detalhes e Pinagem**. 2019. Disponível em: <https://www.fernandok.com/2018/05/nodemcu-esp8266-detalhes-e-pinagem.html>. Acesso em: 19 nov. 2021. 75

GLOBO.COM. **Quase 40% da água potável no Brasil é desperdiçada, aponta levantamento do Instituto Trata Brasil**. 2021. Disponível em: <https://g1.globo.com>. Acesso em: 20 set. 2020. 11

IBM. **Conhecendo o MQTT**. 2019. Disponível em: <https://developer.ibm.com/br/articles/iot-mqtt-why-good-for-iot/>. Acesso em: 02 set. 2021. 24

KENZIE.COM.BR. **O que é React?** 2021. Disponível em: <https://kenzie.com.br/blog/react/>. Acesso em: 17 set. 2021. 39

KICAD. **About Kicad**. 2021. Disponível em: <https://www.kicad.org/about/kicad/>. Acesso em: 02 dez. 2021. 35

MICROSYSTEMS, A. **ACS712 - Fully Integrated, Hall-Efect-Based Liner Current Sensor IC with 2.1 kVRMS Isolation and Low-Resistance Current Conductor**. [S.l.]: Allegro, 2020. 33

MNTOLIA.COM. **Mensagens retidas MQTT explicadas**. 2019. Disponível em: <https://mntolia.com/mqtt-retained-messages-explained-example/>. Acesso em: 02 set. 2021. 26

OPENSOURCE. **What is Linux?** 2021. Disponível em: <https://opensource.com/resources/linux>. Acesso em: 02 dez. 2021. 37

SMART SOLUCTIONS FOR HOME. **How to program an ESP8266 - With and Without Arduino**. 2020. Disponível em: <https://smartsolutions4home.com/how-to-program-esp8266/>. Acesso em: 21 nov. 2021. 75

SOFTWARELIVRE. **O que é Linux Embarcado**. 2021. Disponível em: <https://softwarelivre.blog.br/2014/05/24/o-que-e-linux-embarcado/>. Acesso em: 01 dez. 2021. 37

UFRJ. **Conhecendo o MQTT**. 2019. Disponível em: <https://www.gta.ufrj.br/ensino/eel878/redes1-2018-1/trabalhos-vf/mqtt/>. Acesso em: 02 set. 2021. 24

UMBLER.COM. **Node.js: quem já utiliza a tecnologia (e aprova)?** 2021. Disponível em: <https://blog.umbler.com/br/node-js-quem-ja-utiliza-a-tecnologia-e-aprova/>. Acesso em: 17 set. 2021. 39

VENTAVOLI, M. R. F. **Redes Wireless**. [S.l.]: Fabíola Ventavoli; 2º edição (19 novembro 2014), 2014. 22

WIRED.COM. **Figma Wants Designers to Collaborate Google-Docs Style**. 2021. Disponível em: <https://www.wired.com/story/figma-updates/>. Acesso em: 17 set. 2021. 38

Anexos

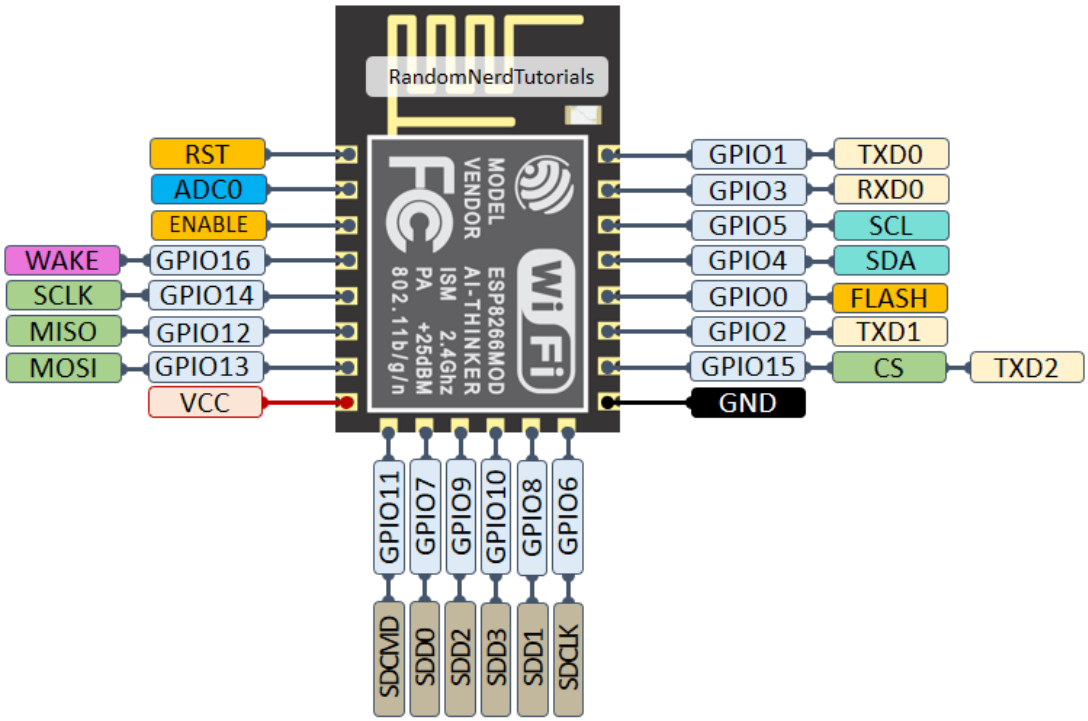
ANEXO A – ESP-12S e ESP-12E

Este anexo abordará a respeito das características dos componentes ESP-12S e ESP-12E. Ambos possuem em seu interior o microcontrolador da Tensilica ESP8266EX. Esse microncontrolador pode ser programado por meio das linguagens C/C++, LUA e Micropython(EMBARCADOS, 2015). O que tornam esses microntroladores interessantes são os recursos de Wi-Fi contidos. O *chip* integra interruptores de antena, *balun* RF, amplificador de potência, filtros e módulos de gerenciamento de energia.

A.1 Características gerais

- 11 GPIO's disponíveis para programação, possuindo barramentos 2C, SPI, UART, entrada ADC e saída PWM;
- Sensor interno de tempertura;
- CPU que opera em 80MHz, com possibilidade de operar em 160MHz (*overclock*);
- Arquitetura RISC 32 bits;
- 32KBytes de RAM para instruções;
- 96KBytes de RAM para dados;
- 64KBytes de ROM para boot;
- Memória Flash SPI Winbond W25Q40BVNIG de 512KBytes;

Figura A.1 – Representação *pinout ESP-12E*



Fonte: Random Tutorials, 2019

A.2 Caracteríticas Específicas

A Tabela A.1 mostra as características de cada pino do ESP8266 (SMART SOLUCTIONS FOR HOME, 2020) em conjunto com um estudo realizado pelo professor Fernando K (FERNANDO K, 2019). Com base nesse estudo, o qual mostra o comportamento dos pinos no momento de *boot*, foi possível encontrar a melhor disposição para associar o microcontrolador com os devidos circuitos de acionamento e sensoriamento.

Tabela A.1 – Detalhamento do funcionamento dos pinos ESP8266

Pino	Direção	Descrição	Ao iniciar
GPIO0	INPUT / OUTPUT	Usado para selecionar o modo de inicialização. Estando no *GND* durante o *boot*, entra no modo de *flash*. Para inicialização normal manter contato com o *VCC* ou deixá-lo em aberto.	*HIGH* na inicialização com oscilações por 120 *ms*.
GPIO1	OUTPUT	Usado como um UART0 TX durante a programação da *flash*. Em modo de operação normal, pode ser utilizado como um GPIO de saída.	*HIGH* na inicialização com oscilações por 80 *ms*.
GPIO2	INPUT / OUTPUT	De uso geral. Geralmente conectado à um *LED*. Não permite o *boot* se, na operação, for colocado em nível lógico baixo.	*HIGH* na inicialização com oscilações por 80 *ms*.
GPIO3	INPUT	Usado como um UART0 RX durante a programação da *flash*. Em modo de operação normal, pode ser utilizado como um GPIO de saída.	Permanece em nível lógico alto durante o *boot*.
GPIO4	INPUT /OUTPUT	Frequentemente usado como *SDA* para *I2C* (qualquer outro pino também pode ser usado). No modo de operação normal, ele pode ser usado como um GPIO clássico.	Permanece em nível lógico baixo durante o *boot*.
GPIO5	INPUT /OUTPUT	Frequentemente usado como *SCL* para *I2C* (qualquer outro pino também pode ser usado). No modo de operação normal, ele pode ser usado como um GPIO clássico.	Permanece em nível lógico baixo durante o *boot*.
GPIO6	NÃO RECOMENDADO	É utilizado para conexão como o pino de *clock* do cartão *SD*	*LOW* na inicialização com oscilações por 500 *ms*.
GPIO7	NÃO RECOMENDADO	Utilizado para transferência de dados com o cartão *SD*	Sempre oscilando.

Continua na próxima página

Tabela A.1 – *Continuação da tabela*

Pino	Direção	Descrição	Ao iniciar
GPIO8	NÃO RECOMENDADO	Utilizado para transferência de dados com o cartão *SD*	*LOW* na inicialização com oscilações por 400 *ms*.
GPIO9	NÃO RECOMENDADO	Utilizado para transferência de dados com o cartão *SD*	Permanece em nível lógico alto durante o *boot*.
GPIO10	NÃO RECOMENDADO	Utilizado para transferência de dados com o cartão *SD*	*LOW* na inicialização com oscilações por 400 *ms*.
GPIO11	NÃO RECOMENDADO	Utilizado para transferência de dados com o cartão *SD*	*LOW* na inicialização com oscilações por 500 *ms*.
GPIO12	INPUT / OUTPUT	Pode ser usado como um pino MISO para a interface HSPI (*High Speed Parallel Interface*). Pode ser um GPIO clássico.	Permanece em nível lógico alto durante o *boot*.
GPIO13	INPUT / OUTPUT	Pode ser usado como um pino MOSI para a interface HSPI (*High Speed Parallel Interface*). Pode ser um GPIO clássico.	Permanece em nível lógico alto durante o *boot*.
GPIO14	INPUT / OUTPUT	Pode ser usado como um pino CLK para a interface HSPI (*High Speed Parallel Interface*). Pode ser um GPIO clássico.	Permanece em nível lógico alto durante o *boot*.
GPIO15	INPUT / OUTPUT	Pode ser usado como um pino CS para a interface HSPI (*High Speed Parallel Interface*). Pode ser um GPIO clássico.	Permanece em nível lógico alto durante o *boot*.
GPIO16	INPUT / OUTPUT	Pode ser usado como *wake up* do modo de *deep sleep* ou como GPIO clássico.	Permanece em nível lógico alto durante o *boot*.
ADC0	INPUT	Conversor analógico digital com 10 *bits* de precisão com variação de *0V - 1V*	Permanece em nível lógico baixo durante o *boot*.
CH_EN	Não aplicável	Pino de ativação do chip: se HIGH o chip funciona normalmente, se LOW entra em modo de baixo consumo	Não apresenta variações

Fim da tabela

ANEXO B – Raspberry Pi Zero W

Figura B.1 – Especificação da Raspberry Pi Zero W retirada do *datasheet*

Specification

Form factor:	65mm × 30mm
Processor:	Broadcom BCM2710A1, quad-core 64-bit SoC (Arm Cortex-A53 @ 1GHz)
Memory:	512MB LPDDR2
Connectivity:	• 2.4GHz IEEE 802.11b/g/n wireless LAN, Bluetooth 4.2, BLE, onboard antenna • 1 × USB 2.0 interface with OTG • HAT-compatible 40-pin I/O header footprint • microSD card slot • Mini HDMI port • CSI-2 camera connector
Video:	• HDMI interface • Composite video
Multimedia:	• H.264, MPEG-4 decode (1080p30) • H.264 encode (1080p30) • OpenGL ES 1.1, 2.0 graphics
Input power:	5V DC 2.5A
Operating temperature:	-20°C to +70°C
Production lifetime:	Raspberry Pi Zero 2 W will remain in production until at least January 2028
Compliance:	For a full list of local and regional product approvals, please visit pip.raspberrypi.com

Fonte: datasheets.raspberrypi.com, 2021

ANEXO C – Informações do transistor IRF1404

Figura C.1 – Recorte do *datasheet* IRF1404.

IRF1404Z/S/LPbF

International
IOR Rectifier

Electrical Characteristics @ T_J = 25°C (unless otherwise specified)

	Parameter	Min.	Typ.	Max.	Units	Conditions
$V_{(BR)DSS}$	Drain-to-Source Breakdown Voltage	40	—	—	V	V_{GS} = 0V, I_D = 250µA
$\Delta V_{(BR)DSS}/\Delta T_J$	Breakdown Voltage Temp. Coefficient	—	0.033	—	V/°C	Reference to 25°C, I_D = 1mA
$R_{DS(on)}$	Static Drain-to-Source On-Resistance	—	2.7	3.7	mΩ	V_{GS} = 10V, I_D = 75A ②**
$V_{GS(th)}$	Gate Threshold Voltage	2.0	—	4.0	V	V_{DS} = V_{GS}, I_D = 150µA
gfs	Forward Transconductance	170	—	—	V	V_{DS} = 25V, I_D = 75A**
I_{DSS}	Drain-to-Source Leakage Current	—	—	20	µA	V_{DS} = 40V, V_{GS} = 0V
		—	—	250		V_{DS} = 40V, V_{GS} = 0V, T_J = 125°C
I_{GSS}	Gate-to-Source Forward Leakage	—	—	200	nA	V_{GS} = 20V
	Gate-to-Source Reverse Leakage	—	—	-200		V_{GS} = -20V
Q_g	Total Gate Charge	—	100	150		I_D = 75A**
Q_{gs}	Gate-to-Source Charge	—	31	—	nC	V_{DS} = 32V
Q_{gd}	Gate-to-Drain ("Miller") Charge	—	42	—		V_{GS} = 10V ②
$t_{d(on)}$	Turn-On Delay Time	—	18	—		V_{DD} = 20V
t_r	Rise Time	—	110	—		I_D = 75A**
$t_{d(off)}$	Turn-Off Delay Time	—	36	—	ns	R_G = 3.0 Ω
t_f	Fall Time	—	58	—		V_{GS} = 10V ②
L_D	Internal Drain Inductance	—	4.5	—		Between lead,
					nH	6mm (0.25in.)
L_S	Internal Source Inductance	—	7.5	—		from package
						and center of die contact
C_{iss}	Input Capacitance	—	4340	—		V_{GS} = 0V
C_{oss}	Output Capacitance	—	1030	—		V_{DS} = 25V
C_{rss}	Reverse Transfer Capacitance	—	550	—	pF	f = 1.0MHz
C_{oss}	Output Capacitance	—	3300	—		V_{GS} = 0V, V_{DS} = 1.0V, f = 1.0MHz
C_{oss}	Output Capacitance	—	920	—		V_{GS} = 0V, V_{DS} = 32V, f = 1.0MHz
C_{oss} eff.	Effective Output Capacitance	—	1350	—		V_{GS} = 0V, V_{DS} = 0V to 32V ④

Source-Drain Ratings and Characteristics

	Parameter	Min.	Typ.	Max.	Units	Conditions
I_S	Continuous Source Current	—	—	120⑩		MOSFET symbol
	(Body Diode)				A	showing the
I_{SM}	Pulsed Source Current	—	—	750		integral reverse
	(Body Diode) ①					p-n junction diode.
V_{SD}	Diode Forward Voltage	—	—	1.3	V	T_J = 25°C, I_S = 75A**, V_{GS} = 0V ③
t_{rr}	Reverse Recovery Time	—	28	42	ns	T_J = 25°C, I_F = 75A**, V_{DD} = 20V
Q_{rr}	Reverse Recovery Charge	—	34	51	nC	di/dt = 100A/µs ③
t_{on}	Forward Turn-On Time	Intrinsic turn-on time is negligible (turn-on is dominated by LS+LD)				

Fonte: alldatasheet.com