UNIVERSIDADE FEDERAL DO PAMPA

GERSON EVANDRO DE OLIVEIRA SENA

MEDIDOR DE CONSUMO DE ENERGIA ELÉTRICA COM ACESSO LOCAL E REMOTO USANDO PLATAFORMA ESP8266

Alegrete
2018

GERSON EVANDRO DE OLIVEIRA SENA

MEDIDOR DE CONSUMO DE ENERGIA ELÉTRICA COM ACESSO LOCAL E REMOTO USANDO PLATAFORMA ESP8266

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia Elétrica.

Orientador: Jumar Luís Russi

Alegrete
2018

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

S474m Sena, Gerson Evandro de Oliveira
MEDIDOR DE CONSUMO DE ENERGIA ELÉTRICA COM ACESSO LOCAL E REMOTO USANDO PLATAFORMA ESP8266 / Gerson Evandro de Oliveira Sena.
76 p.

Trabalho de Conclusão de Curso(Graduação)-- Universidade Federal do Pampa, ENGENHARIA ELÉTRICA, 2018.
"Orientação: Jumar Luís Russi Russi".

1. Medidor de energia. 2. ESP8266. 3. Datalogger. 4. Protocolo MQTT. I. Título.

GERSON EVANDRO DE OLIVEIRA SENA

MEDIDOR DE CONSUMO DE ENERGIA ELÉTRICA COM ACESSO LOCAL E REMOTO USANDO PLATAFORMA ESP8266

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia Elétrica.

Trabalho de Conclusão de Curso defendido e aprovado em: ___/ ___/ 2018.

Banca examinadora:

_____

Prof. Dr. Jumar Luís Russi
Orientador
(UNIPAMPA)

_____

Prof. Me. Lucas Compassi Severo
(UNIPAMPA)

_____

Prof. Dr. Alessandro Botti Benevides
(UNIPAMPA)

A minha Suelen e a nossa pequena Líria, por darem propósito a esta conquista. E aos meus pais, Vera e Ernande, pela minha existência e por, de alguma forma, eu ter tomado as decisões que tomei até aqui.

AGRADECIMENTO

Ao Prof. Dr. Jumar Luís Russi, pela paciência e apoio na orientação deste trabalho e em outros momentos.

Aos professores que souberam ser educados, cordiais, justos e profissionais enquanto ministravam aulas e avaliações em toda minha trajetória acadêmica. Em especial aqueles que realmente tinham prazer em ensinar, que contagiaram com seu conhecimento e sua boa vontade para com os alunos em dificuldades, como eu.

A todos os colegas de curso e percurso que ajudaram a entender conceitos, resolver problemas, entregar trabalhos no prazo, concluir projetos, que animaram, encorajaram, motivaram a persistir, entenderam as dificuldades, ou seja, que estenderam a mão e permitiram. E cujos nomes não caberiam numa única página.

“A grandeza não é onde permanecemos, mas em qual direção estamos nos movendo. Devemos navegar algumas vezes com o vento e outras vezes contra ele, mas devemos navegar, e não ficar à deriva, e nem ancorados”.

Oliver Wendell Holmes Sr.

RESUMO

O consumo de energia elétrica e o controle deste consumo éum tema muito importante no contexto atual brasileiro, já que os valores cobrado e taxado sobre esta energia são alguns dos mais altos a nível global, onde um terço do consumo consta de consumidores residenciais. O cenário se torna ainda mais crítico ao se notar a ausência e o difícil (ou inexistente) acesso a mecanismos de auditagem e controle do consumo pelo cliente, o que colabora negativamente para criação de uma cultura de economia e bom uso de energia elétrica. Como possibilidade de resposta a estas questões, este trabalho busca propor a arquitetura de um protótipo de medidor de energia elétrica. A iniciativa visa pesquisar, avaliar e propor um sistema de baixo custo, acessível e viável dos pontos de vista econômicos e práticos para clientes residenciais. Com isso espera-se motivar a pesquisa e desenvolvimento de medidores que não demandem complexidade nem custo acima da realidade de consumo de clientes residenciais. O sistema a ser apresentado consta de um medidor de tensão e corrente elétricas senoidais (corrente alternada), cujo núcleo é um módulo de desenvolvimento NodeMCU Lolin (equipado com um microcontrolador ESP8266-12E), e aproveitando os recursos nativos à plataforma Arduino. As funcionalidades do protótipo são as medições já citadas, o cálculo da potência ativa em tempo real, o armazenamento do consumo de energia, o registro de todos estes dados em uma memória temporal recuperável (datalogger) e também transmissão destes dados com um protocolo Message Queue Telemetry Transport (MQTT) para ser recuperado por aplicativos WEB ou de telefones celulares, com conexão via Internet.

Palavras-Chave: Medidor de Energia, ESP8266, Datalogger, protocolo MQTT.

ABSTRACT

Few topics are as current as the need and consumption of electricity. And within the economic context, some of the issues addressed concern the quality and waste of this energy. Meanwhile, we live in one of the countries where the values charged for this energy are some of the highest globally, where a third of consumption consists of residential consumers. The scenario becomes even more critical when one notices the absence and the difficult (or nonexistent) access to mechanisms of auditing and control of consumption by the customer, which contributes negatively to the creation of a culture of economy and good use of electric energy. As a possible answer to these questions, this work seeks to propose the way of building a prototype of an power meter. The initiative aims to research, evaluate and propose a low cost, affordable and affordable system from the economic and practical points of view for residential customers. This is expected to motivate the research and development of meters that do not demand complexity or cost above the consumption reality of residential customers. The system to be presented consists of a sine wave electric current and voltage meter, whose core is a NodeMCU Lolin module equipped with an ESP8266-12E, built with features native to the Arduino platform. The prototype features are the aforementioned measurements, real-time active power calculation, energy consumption storage, recording of all this data on a memory card (datalogger) and also transmission of this data with an MQTT protocol (to be retrieved by WEB applications or cell phones) via the Internet.

Keywords: Power Meter, ESP8266, Datalogger, MQTT protocol

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Verificar a necessidade desta seção...

SUMÁRIO

1. INTRODUÇÃO

É perceptível que a energia elétrica faz parte do nosso cotidiano. Desde sua geração até seu consumo final. Muitas tarefas, dispositivos e recursos são demandados. Neste estudo, a perspectiva do consumidor é abordada.

Muitas atividades, das mais simples às mais sofisticadas, demandam consumo de energia elétrica, que por sua vez demandam uma crescente necessidade de produção. A demanda crescente é uma realidade, inclusive, dos pequenos consumidores.

Dados oficiais da Agência Nacional de Energia Elétrica (ANEEL) e do Ministério de Minas e Energia (MME) indicam que o consumo nacional residencial de energia elétrica já era próximo a 30% do total gerado (ANEEL, 2002), e mesmo tendo havido uma queda nesse percentual em meados de 2015 o consumo por esse grupo consumidor era próximo a 22% (MME, 2015). Também dados da Empresa de Pesquisa Energética (EPE) mostram um crescimento médio próximo a 2,5% a.a. (EPE, 2017) mesmo somando-se vários períodos de decrescimento e perdas econômicas.

Entretanto, a solução para o crescimento do consumo não reside apenas no crescimento da geração, mesmo este sendo necessário e estratégico. Segundo a Associação Brasileira das Empresas de Serviços de Conservação de Energia (Abesco) cerca de R$ 12,6 bi são o saldo negativo do desperdício de energia elétrica no Brasil, sendo que a principal fatia, em torno de R$ 5,51 bi, são do tipo de consumidor residencial (ABESCO, 2015; CUNHA, 2015). São estimadas perdas por desperdício de “460 mil GWh em quatro anos, suficientes para suprir a demanda do país em um ano” (GUADANIN, 2015).

É então perceptível que ações junto aos consumidores, especialmente aos pequenos, poderão vir a ter contribuição significativa, tanto na economia como na expansão dos serviços de fornecimento de energia elétrica. Também algumas hipóteses podem ser levantadas de que pode haver carência de ações políticas, mudanças de hábitos de consumo e de instrumentação e tecnologias acessíveis que permitiriam auxiliar nessa redução do desperdício. As primeiras hipóteses não são tratadas aqui.

Focando na questão de instrumentação e tecnologias uma opção seria de medidores inteligentes. Ao pesquisar-se sobre o assunto nota-se que já é uma pauta e que houve algumas iniciativas, inclusive por parte das distribuidoras e concessionárias de energia elétrica, mas pelo que se nota em alguns periódicos (HAYASHI, 2018; NUWER, 2015) ao que parece o foco não é pela transparência dos dados em tempo real nas residências, mas para controle de tarifação dos usuários. Há instrumentos no mercado especialmente focados nas opções por

Tarifa Branca e Geração Distribuída, mas quase nenhum produto acessível ao consumidor que lhe permita apenas monitorar seu consumo, sendo o caso mais próximo uma iniciativa AES Eletropaulo e WEG (WEG, 2014).

Agregando a este contexto, há ainda o surgimento crescente na última década de novas ferramentas e recursos voltados ao desenvolvimento de baixo custo para sistemas embarcados, agregar 'inteligência' e comunicação para aparelhos e "coisas" do cotidiano das pessoas, justamente denominado Internet das Coisas (ou mais popularmente *IoT*, do inglês *Internet of Things*). Este é justamente um termo cunhado que vem agregando uma quantidade enorme de equipamentos à internet, não mais se limitando aos computadores e mais recentemente aos telefones celulares.

Hoje é possível encontrar no mercado — ainda mais vastamente no cenário internacional — uma gama enorme de módulos (microcomputadores, microcontroladores, kits de desenvolvimento, etc.) com maior poder de processamento que muitos computadores antigos. O custo é de poucas dezenas ou unidades de dólar e o foco voltado para facilidade de integração e aprendizado, também fez com que a popularização de tais recursos tivesse não apenas grande aceitação como também maior democratização deste tipo de conhecimento.

Este trabalho procura, portanto, propor um medidor de consumo de energia elétrica de uso não industrial, aproveitando os recursos de desenvolvimento citados, dentro da filosofia Código Aberto[1]. Que permita ao usuário ter livre acesso às informações sobre seu consumo de energia elétrica, podendo inclusive ser alterado para agregar mais funções.

Protótipos dentro deste tema proposto já são encontrados na internet bem como em pesquisas acadêmicas. O que se pretende mostrar é a aquisição de habilidades e conhecimentos para a criação de um dispositivo nos moldes: acessibilidade, confiabilidade, baixo custo, transparência, modularidade.

Assim, as ideias de aplicação recaíram sobre módulos de desenvolvimento e circuitos pré-acabados utilizando microcontroladores em módulos de baixo custo. Inicialmente um projeto funcional foi concebido em plataforma Arduino (UNO R3), mas logo foi modificado e expandido para plataforma baseada em módulos ESP8266 (neste caso um NodeMCU v3,

[1] O "**Código aberto**, ou *Open Source* em inglês, é um modelo de desenvolvimento que promove um licenciamento livre para o design ou esquematização de um produto, e a redistribuição universal desse design ou esquema, dando a possibilidade para que qualquer um consulte, examine ou modifique o produto". Fonte: Wikipédia (2013), < https://pt.wikipedia.org/wiki/C%C3%B3digo_aberto>.

Lolin), que é o núcleo por trás de todo o projeto. O NodeMCU agrega a compatibilidade com o que já foi desenvolvido para Arduino e ainda a conexão Wi-Fi integrada ao sistema.

Para leitura dos sinais de tensão e corrente elétricas foram buscadas soluções em circuitos integrados dedicados, recaindo a tarefa sobre um módulo genérico da fabricante Peacfair, o modelo PZEM-004T, cujo núcleo do tipo 'sistema-em-um-chip' (*SoC – System-on-a-Chip*) SD3004 exclusivo, produzido pela SDICMicro (ambas empresas chinesas).

Com relação à comunicação, neste trabalho optou-se por um protocolo 'máquina-à-máquina' do tipo Transporte por Telemetria de Filas de Mensagens (*MQTT – Message Queue Telemetry Transport*). Com base nesse protocolo há vários servidores livres (ou pagos), baseados em computação em 'nuvem', de relativa facilidade de configuração, alguns disponibilizando aplicativos para dispositivos móveis (smartphones, tablets, etc.). O uso de bibliotecas para uso em sistemas embarcados (como ESP8266, Arduino, RaspBerry Pi, etc.) também foi aproveitado.

1.1. JUSTIFICATIVA

O consumidor tem (ou deveria ter) o direito de saber exatamente o que acontece em termos de consumo de energia em sua residência (ou aparelhos). Além do viés ético e econômico da transparência, há questões como de identificar anormalidades nas instalações elétricas (fugas, perdas, faltas), especialmente quando o usuário não se encontra no local (em horário de trabalho, férias, etc.).

A transparência do consumo se torna ainda mais relevante dadas as novas formas de tarifação que vem sendo aplicadas ou estudadas para os clientes residenciais (como as bandeiras tarifárias: tarifa branca), demandas pré-contratadas (como ocorre com a telefonia celular), sistemas de medição bidirecionais (clientes aplicando geração própria: fotovoltaicas e eólicas).

Ainda, os poucos medidores de consumo existentes no mercado nacional (a grande maioria importados) não agregam funções de ação remota, sejam de simples leitura ou mesmo de ações de telecomando para o usuário.

Havia o desejo de aprendizado (somado à oportunidade e acessibilidade), então, de desenvolver um sistema de medição nos moldes Código Aberto.

1.2. MOTIVAÇÃO

A área de embarcados e Internet das Coisas (*IoT – Internet of Things*) são movimentos que tomam cada vez mais força, popularidade e aporte de recursos e investimentos. Questões com relação às fontes de energia, automação de processos, solução

de problemas em termos sociais, de economia ou ambientais, também são temas sempre em voga.

A ideia de criar algo que tivesse apelo e aplicação pragmática em algum desses quesitos (ou similares), que também fosse instigante já é motivadora por si só, seja nos esboços de projetos e até mesmo de ideias de negócios ou produtos. Uma dessas ideias era a necessidade de medição de consumo de energia elétrica de modo transparente e acessível, um produto que também poderia agregar alguma automação ou ser a porta de entrada para outras interconexões com aplicações diversas (em ambiente doméstico, por exemplo).

A filosofia de Código Aberto permite abertura para ensaios iniciais com fins didáticos e comerciais. Iniciar por plataformas prontas (como Arduino, Raspberry Pi, ESP8266, teensy, etc.) resolve vários problemas que acometem iniciantes (ou mesmo profissionais experientes), especialmente se fatores como tempo e recursos a serem investidos não são abundantes. É de se concordar que como produto acabado estas não parecem ter tanto propósito econômico muito maior que o didático.

Por trás de um módulo de desenvolvimento existe alto grau de complexidade e aprendizado (explícitos ou implícitos). Seu manuseio promove aprendizado, que incentiva e fortalece o desejo de se aprofundar-se mais nos assuntos. A linguagem de máquina, a arquitetura do hardware, o tratamento de sinais, os algoritmos são alguns dos pontos que vão tomando corpo mais complexo até um conhecimento mais sólido e pleno.

Este projeto, com todas as suas partes, detalhes e complexidades, não teria sido possível no curto espaço de tempo e com os poucos recursos disponíveis sem o acesso ao Open Source. Desenvolver cada um dos módulos e soluções exigiria não apenas equipes multidisciplinares, como muito mais recursos e tempo disponíveis.

O que será percebido é que com poucos recursos se teve acesso a conhecimentos sobre tendências comerciais e tecnológicas, hardware, protocolos, circuitos dedicados, técnicas de comunicação, processamento de sinais, além de permitir criar uma relação com a comunidade colaborativa (fóruns, sites, repositórios) comum a este tipo de projeto.

1.3. OBJETIVO GERAL

Apresentar uma solução para leituras das grandezas básicas de sinais e de consumo de energia elétrica de um equipamento cujos valores estarão disponíveis ao usuário final por pelo menos três meios de acesso:

- Via internet (on-line): via dispositivo móvel ou navegador;
- Arquivo (off-line): gravação de arquivo em um cartão de memória (datalogger);

- Local de medição: display digital.

1.3.1. Objetivo específico

O protótipo deve contemplar:

- Adquirir, armazenar e permitir a visualização dos valores de tensão, corrente, potência ativa instantânea e consumo de energia elétrica;
- Permitir fácil conexão ao padrão de medição residencial (ou outra carga), critério de 'plug-and-play';
- Ter um aplicativo com interface intuitiva em plataforma Android ou IOS, critério de multiplataforma;
- Possuir precisão equiparada a instrumentos comerciais, critério de acuricidade;
- Possuir back-up de dados e de alimentação dos módulos, critério de confiabilidade;
- Deve ainda permitir a inserção ou modificação do sistema sem custos significativos, agregando assim novas funções;
- Deve propiciar se possível, na sua construção, ideias e conhecimentos iniciais para desenvolvimento de um produto ou serviço comercial, em hardware mais robusto, simples e viável comercialmente.

1.4. ORGANIZAÇÃO DO TRABALHO

Após essa breve introdução de defesa do tema e título o restante deste trabalho aqui apresentado constará das seguintes partes:

1.4.1. Revisão Bibliográfica.

Esta seção inclui grande parte do trabalho, podendo ser observados:

Metodologia: que descreve como foi pensado o projeto e os passos seguidos até sua conclusão. Detalha o que foi realizado, na sequência em que ocorreu.

Conceitos, Hardware e Código:

- Trata cada questão a ser solucionada e da resposta dada;
- Faz análise de alguns itens usados;
- Delineia os produtos, partes e peças escolhidos, discorrendo sobre o que foi aprendido sobre;
- Em alguns pontos sinaliza outras possibilidades, quando pertinente.

1.4.2. Desenvolvimento do Protótipo

Nesta seção, a metodologia e aplicação dos conhecimentos gerados na seção de revisão bibliográfica são apresentadas ao leitor.

1.4.3. Resultados

Nesta seção procura-se mostrar os pontos onde se obteve sucesso na proposta e execução do protótipo. É inclusive onde os dados obtidos através do protótipo são comparados com valores lidos por outros instrumentos comerciais.

1.4.4. Conclusões

Última seção textual do trabalho trata de modo sucinto os fatos e considerações finais na conclusão do projeto. Cita ainda melhorias e possíveis caminhos que podem ser tomados para trabalhos futuros.

1.4.5. Elementos Pós-Textuais.

Contém material suplementar e corroborativo do trabalho:

- Referências Bibliográficas citadas;
- Anexos com esquemas, tutoriais, códigos e outros materiais necessários para maior entendimento, mas que não caberiam à fluidez do trabalho escrito.

2. REVISÃO BIBLIOGRÁFICA

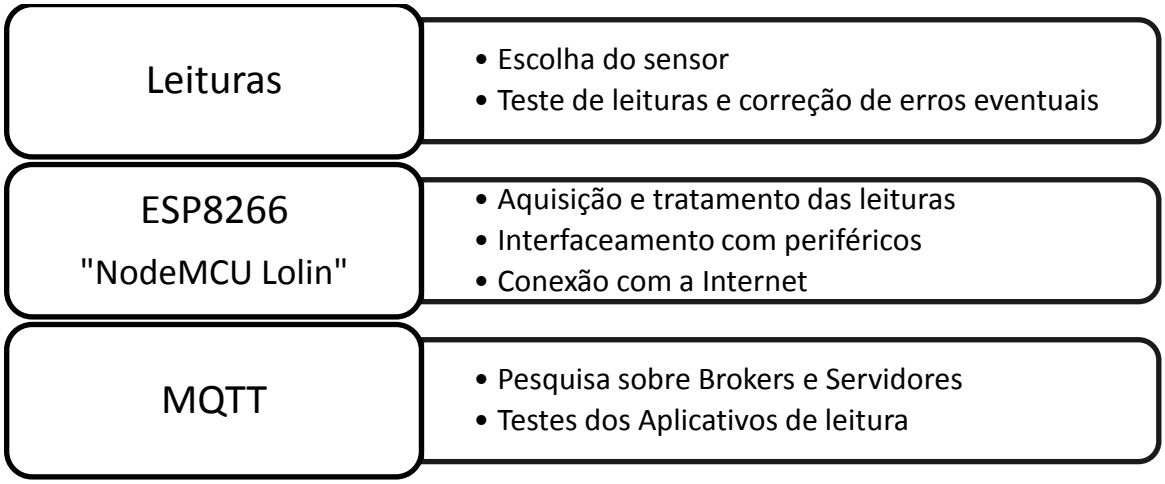
Nesta seção apresentamos apenas os principais conhecimentos teóricos observados no trabalho (por questões de espaço e fluidez textual), referenciados quando necessário. Serão também apresentadas as informações técnicas que forem pertinentes. Algumas seções mais tutoriais ou pormenorizadas foram movidas à seção de anexos.

2.1. METODOLOGIA E DETALHES DO PROJETO

O que se buscou neste trabalho foi conseguir desenvolvê-lo conforme os objetivos inicialmente propostos, com ênfase em modularidade e códigos abertos. Problemas e soluções propostas foram citados ao longo desta seção.

O diagrama de blocos da Figura 1 ilustra as frentes de trabalho.

Figura 1 – Resumo da metodologia aplicada neste trabalho.

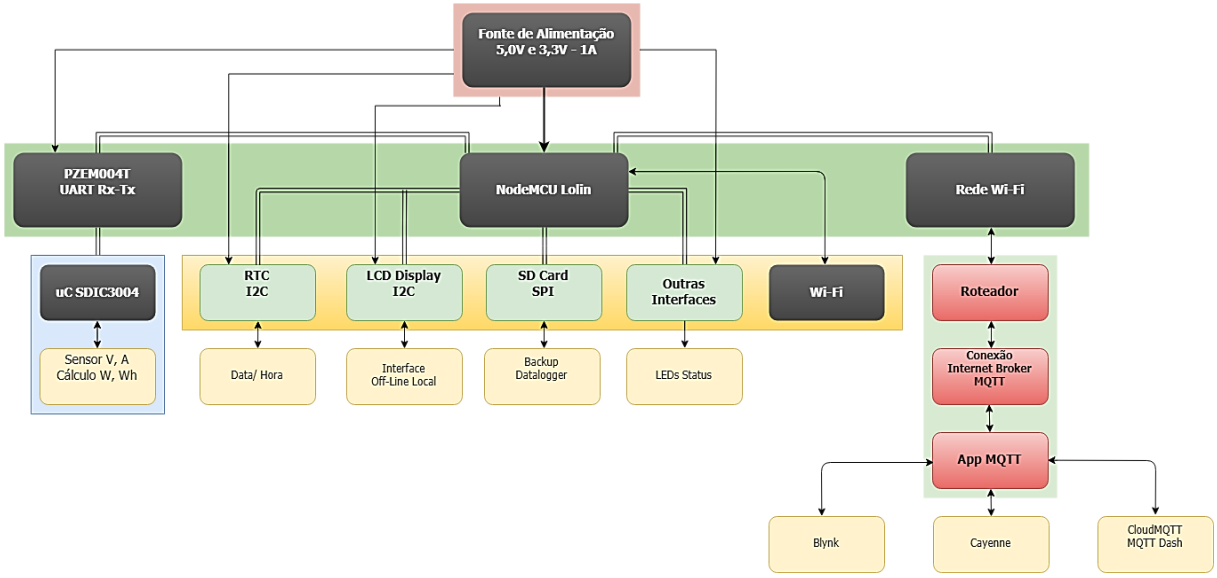


Fonte: Autor.

O esboço da metodologia permitiu organizar cronogramas de pesquisa por ordem de prioridades.

Já o diagrama de blocos de hardware da Figura 2 visava organizar a bancada de trabalho e dar a devida atenção a cada solução que seria necessária:

Figura 2 – Diagrama em blocos das partes componentes para desenvolvimento do sistema.



Fonte: Autor.

Com auxílio deste diagrama foi possível determinar em que módulo trabalhar prioritariamente e também quais deles testar em separado, resolvendo os problemas de cada parte assim que fossem surgindo. Os blocos acima foram resolvidos primeiramente, para garantir que se algo mais não tivesse resultados positivos, ainda assim haveria um projeto mínimo viável.

2.2. Internet das Coisas, Código Aberto e Hardware Aberto

Mostrou-se um trabalho bem difícil definir em termos simples, concisos e abrangentes o que seria Internet das Coisas (referida a partir desse ponto do trabalho apenas como *IoT*, da siga em inglês). O termo mais simples seria o de conectar coisas (e não pessoas) à Internet, embora como lembrado por (JAVED, 2017, p. 14) seja mais do que isso.

Uma pesquisa sobre o tema, livros, artigos e outros formatos de publicações sobre o tema faz perceber que embora a prática de dispositivos conectados em redes de comunicação não seja uma coisa nova, a *IoT* já parece ser algo desta última década. Relatório da (CISCO, 2011, pp. 2-3) trata como algo recente, "uma evolução da internet". De fato, a maioria das publicações são posteriores a 2010[2]. Segundo o mesmo artigo é estimado que a projeção global de crescimento da *IoT* seja de 50 bilhões de dispositivos conectados até 2020. O que faz pensar que as projeções de faturamento e soluções para o seu desenvolvimento não serão pequenas.

[2] Pesquisa realizada pela quantidade de buscas pelo termo em inglês de *IoT* no *Google Analytics*, por ano, desde 2000.

(MANCINI, 2017) ainda nos lembra de um leque de novas possibilidades de aplicações como, por exemplo, as cidades inteligentes (*smart cities*); a saúde (*smart healthcare*); as casas inteligentes (*smart home*) e desafios que virão (regulamentações, segurança, padronizações). Que demandarão grande número de oportunidades para pesquisas, projetos e negócios.

2.3. Código e Hardware Aberto

O Código Aberto trata-se de uma definição que abrangia programas de computador cujos códigos poderiam ser acessados e modificados livremente sem que precisasse pagar taxas aos desenvolvedores e nem haveria problemas com quebras de patentes ou plágio (contanto que os devidos créditos fossem dados). E em geral se popularizou pelos softwares livres, que muitas vezes são confundidos apenas por 'grátis', que são na verdade coisas distintas (ALECRIN, 2013).

O movimento se expandiu algum tempo depois para o conceito de Open Hardware:

> "O conceito de Open Hardware é muito parecido com o conceito de software livre, popularizado pelo sistema operacional Linux, cujo código fonte é aberto, ou seja, você pode baixar o código, fazer as suas próprias modificações e criar um Linux customizado. Com o **Open Hardware** acontece praticamente a mesma coisa. São circuitos eletrônicos ou hardware de computador que podem ser copiados livremente, [todo projeto estará disponível]. Geralmente o desenvolvedor não cobra nenhuma taxa de licença, mas em alguns casos é exigido que o nome dele seja incluído nos créditos do projeto final. Também podendo exigir que qualquer projeto baseado em seu trabalho seja distribuído no esquema de Open Hardware." (THOMSEN, 2014).

2.4. Resumo dos Protocolos e Padrões Atualmente comuns para IoT

Como é possível perceber, a *IoT* prescinde grandemente de comunicação (entre módulos, servidores, dispositivos, etc.). Ainda que haja desafios e se prospecte ainda novas tecnologias como resposta ou recursos a serem melhorados, a *IoT* faz ainda grande uso dos sistemas de comunicação populares ou de sistemas embarcados já existentes, listados na Tabela 1.

Tabela 1: Comparação entre Tecnologias de Comunicação Comuns em IoT.

Protocolo	Alcance (m)	Frequência	Taxa (bps)	IPv6	Topologia
Ethernet	100/ 2000 m	N/A	10 Gbps	Sim	Variada
Wi-Fi	50 m	2,4/ 5 GHz	1300 Mbps	Sim	Estrela
BLE	80 m	2,4 GHz	1 Mbps	Sim	Estrela/ Mesh
ZigBee	100 m	915MHz/ 2,4 GHz	250 kbps	Sim	Estrela/ Mesh
3G/ 4G	35/ 200 km	1900/ 2100/ 2500 MHz	1/ 10 Mbps	Sim	Estrela

Protocolo	Alcance (m)	Frequência	Taxa (bps)	IPv6	Topologia
SigFox	10/ 50 km	868/ 902 MHz	10 – 1000 bps	-	-
LoRaWAN	2/ 5 km	Sub-GHz	0,3 – 50 kbps	Sim	Estrela

Fonte: Internet das Coisas: da Teoria à Prática (SANTOS, et al).

Um destaque especial para tecnologias recentes é inclusive o surgimento do Protocolo LoRaWAN[3]. No ambiente mais doméstico continuam a prevalecer as populares redes Ethernet e Wi-Fi, com também grande participação de dispositivos 3G/4G graças ao advento dos *smartphones*.

Há ainda uma gama de definições sobre redes e comunicações que precisam ser conhecidas para se trabalhar satisfatoriamente neste tipo de projeto. Algumas delas podem ser listadas aqui, mas seu detalhamento foge do escopo deste trabalho:

- Arquitetura Cliente-Servidor;
- TCP/IP, e suas camadas de aplicação, transporte, rede;
- DHCP e DNS, além do TFTP;
- Redes, Topologias e Infraestrutura;
- Protocolos pensados ou repensados para IoT: ZigBee, 6LoWPAN, WirelessHart, ISA100.11a, LoRaWAN, etc.;
- Outras tecnologias, como *Bluetooth* e IR;
- Além de linguagens de programação específicas para navegadores de internet ou aplicativos de celulares (Java, PHP, HTML, MySQL, Python, etc.) e desenvolvimento dos próprios *firmwares* dos sistemas embarcados (C++, Java, Lua, Assembly, etc.).

Ou seja, dependendo da aplicação que se pretende desenvolver, conhecimentos deste tipo precisarão ser levados em consideração.

2.5. Datalogger

É um aparelho, geralmente computadorizado, que tem por objetivo coletar dados, registrando essas informações para serem transmitidas para outros dispositivos, seja armazenando para download em um computador ou transmitindo a algum servidor remoto. São usados em locais onde a conexão direta a um computador se torna difícil ou onerosa ao projeto ou serviço.

O datalogger tem como principal característica coletar dados horários, em tempo real, registrando o momento em que foram lidos. Estes dados são armazenados em sua

[3] Mais algumas informações no artigo: Conheça a tecnologia LoRa® e o protocolo LoRaWAN™, do Eng. Vidal Pereira da Silva Junior, publicado no site Embarcados em 2016: <https://www.embarcados.com.br/conheca-tecnologia-lora-e-o-protocolo-lorawan/>.

memória para posterior leitura ou transmissão, mas nada impede que paralelamente estes dados também sejam transmitidos em tempo real para outro local remoto.

2.6. Medição de Sinais Elétricos e o Módulo de Medição PZEM004T

A medição de um sinal passa pelo conhecimento de modelos que o descrevem. O conhecimento destes sinais e as características ideais dos sensores a serem empregados é crucial neste tipo de projeto, já que o sensor pode ser considerado o coração de todo o sistema de medição a ser projetado.

O módulo PZEM004T, com um μC SD3004 em seu núcleo, fará essas medições.

Os sinais elétricos principais deste trabalho são tensão e corrente elétricas, e sua forma de onda é aproximadamente senoidal (corrente alternada). As magnitudes serão da ordem de 260 *volts*, 100 *amperes* e 60 *hertz* (tensão, corrente e frequência, respectivamente).

O produto da tensão e corrente, medido diretamente, é a potência elétrica instantânea, como em [1]. Esta é, portanto, a capacidade de trabalho instantâneo.

$$P = \frac{dW}{dt} \qquad [1]$$

Onde: P é potência em *watts*, sendo $W = V.I$, V tensão em *volts* e I a corrente em *amperes*.

Que por tratar-se de medições discretas e já estarem convertidas para valores eficazes (o μC SD3004 já faz a conversão para valores *rms*), pode ser resumida a [2]:

$$P = V.I \qquad [2]$$

A energia consumida (e tarifada) é definida como sendo o trabalho pelo tempo [3]. Logo a variável tempo também precisa ser considerada.

$$w_h = \int_{t_0}^{t} p\, dt \qquad [4]$$

Onde:

Que indica que o somatório das potências consumidas (entregues) em um intervalo de tempo $t\ a\ t_0$ é a energia total do processo. Que também por tratar-se de um acumulador discreto (dentro do μC SD3004)

$$E = P.t \qquad [3]$$

Onde: *Wh* é a energia utilizada em *watt-hora*, *p* a potência instantânea e *t* é a unidade de tempo para registrar o período transcorrido.

Lembrando que a tarifação do consumo de energia elétrica de clientes residenciais corresponde ao somatório da potência ativa instantânea durante um determinado período de tempo.

Com essas considerações iniciais em mãos, é preciso determinar os tipos e características de sensores (ou transdutores) necessários. O sensor basicamente reduz a um valor seguro ao mesmo tempo confiável o valor de uma grandeza, já o transdutor, além disso, converte o sinal lido para outro tipo de sinal (geralmente elétrico) em níveis que permitam a leitura.

Dentre as características a serem analisadas nos sensores, destacam-se[4]:

- *Sensibilidade*: termo que indica qual a menor variação de intensidade da grandeza medida, ou seja, o quão detalhado ou pequeno é o valor que pode ser lido.
- *Faixa Nominal* (ou *Range*): é basicamente a escala de menor a maior valor que o sensor ou instrumento é capaz de detectar.
- *Precisão* (ou *acuracidade*): diferente da sensibilidade é, em termos de percentagem, o quando do valor lido reflete a realidade do valor da grandeza.
- *Linearidade*: dentro da faixa nominal, de que valores a que valores a resposta ao que é lido é sempre igual (margem de erro igual).
- *Histerese*: diz respeito ao modo de resposta do sensor tanto na variação positiva (subida) como negativa (descida) do valor da grandeza do sinal.
- *Tempo de Resposta*: basicamente é qual o limite de menor tempo para o qual o sensor consegue fazer a leitura (o sensor não conseguirá ler valores num intervalo de tempo menor).

Em termos de aquisição de sinais de corrente elétrica os sensores ainda podem ser classificados, entre tantas outras coisas, entre invasivos e não invasivos (quando não precisam alterar ou interferir na estrutura do sistema). Dois exemplos podem ser vistos na Figura 3:

Figura 3 – (a) Sensor de corrente elétrica invasivo ACS712 (b) e sensor de corrente não invasivo SCT013.

(a) (b)

Fonte: www.arduinopedia.com.br

Como se percebe, o sensor não invasivo tem a vantagem de não precisar interromper o fio para medir a corrente elétrica que circula por ele, além de propiciar maior segurança no manuseio e no projeto do hardware.

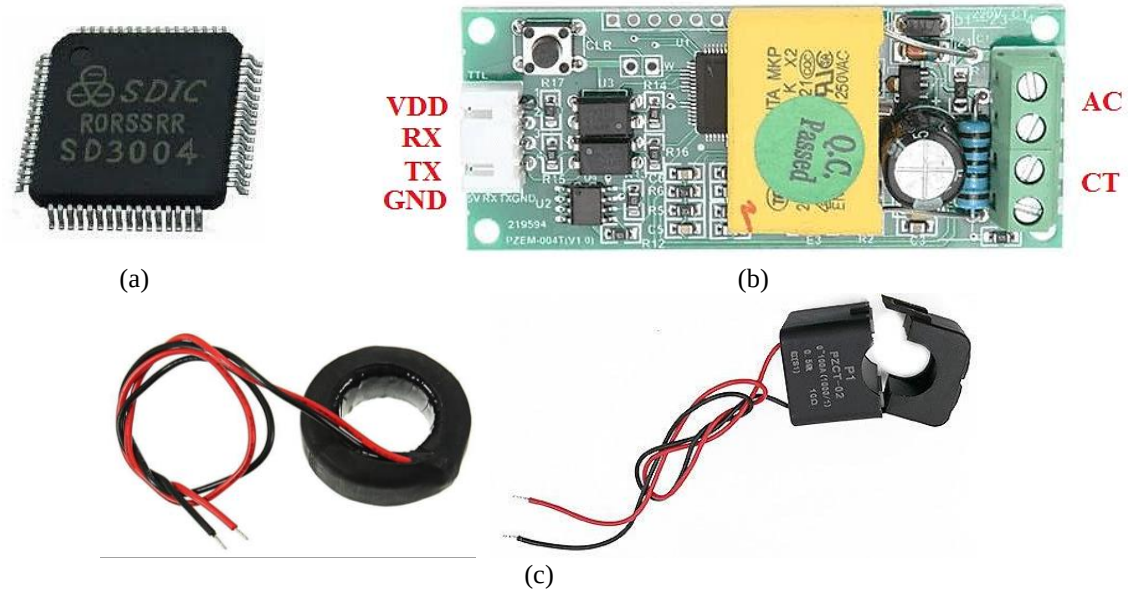
[4] Os termos podem diferir sutilmente em algumas literaturas técnicas.

2.6.1. Módulo PZEM-004T da Peacefair

O módulo PZEM-004T é uma das versões de vários medidores de grandezas elétricas desenvolvido e distribuído pela Peacefair (China). O núcleo do módulo é um SoC SD3004, fabricado pela SDICMICRO[5]. Mas curiosamente, dentro do módulo da Peacefair, muitas das funções do circuito integrado (CI) estão desabilitadas (SDIC, 2012). Entre elas, não permitir e nem fornecer os códigos de acesso (talvez o firmware gravado nele não tenha essa função) ao valor do Fator de Potência (FP), do alarme de limite de leitura e os sinais que habilitam usar um mostrador com display de LED de 7 segmentos (disponíveis em outras versões do produto).

Na Figura 4 consta a aparência do módulo e também do microcontrolador.

Figura 4 – (a) Encapsulamento do Circuito Integrado SD3004, (b) vista superior do módulo medidor de energia PZEM-004T e (c) versões dos TCs disponíveis para o módulo.



Fonte: (Google).

Mas note que ao adquirir o módulo é preciso ter atenção ao tipo de TC (ou CT) que acompanha, se será do tipo anel sólido ou alicate. O módulo não funciona sem esse TC.

Recomenda-se adquirir o módulo com a versão de TC tipo alicate, pois facilita a passagem do fio pelo seu interior, a menos que o módulo seja instalado dentro de um circuito fixo que não necessite mover o cabo sob medição depois de instalado.

O TC que acompanha o módulo é calibrado para este, a tentativa de usar outros TCs causou leituras erradas.

---

[5] http://www.sdicmicro.com/index.html

O módulo da Peacefair proporciona fácil conexão e também isolamento da rede elétrica em medição (pelo uso de opto-acopladores em sua saída).

A Tabela 2 lista as características elétricas deste módulo.

Tabela 2: Características elétricas de aplicação do PZEM-004T.

Grandeza	Range do PZEM	Sensibilidade (Dígitos Visíveis)
Tensão	80,0 – 260,0 V	± 0,1 V
Corrente	0,00 – 99,99 A	± 0,01A
Potência	0 – 22 kW	± 1 W (de 0 a 9999W), ou ± 10 W (de 10000 a 22000W)[6]
Energia	0 – 9999 kWh	± 1 Wh
Frequência	50/ 60 Hz	Não Informada
Precisão AD	μC SD3004	16 bits

Fonte: Peacefair

Algumas das funcionalidades do módulo são:

- Medição das grandezas elétricas de tensão e corrente com saída RMS.
- Cálculo interno de potência ativa e registrador (com memória) de energia consumida.
- O circuito do módulo é alimentado com nível de 5V, comum à maioria dos microcontroladores.
- Função de exposição de tensão, corrente, potência ativa, energia consumida que podem ser lidas por comunicação serial TTL (UART e I2C interfaces), podendo se comunicar com diversos dispositivos. Com um barramento de 16 bits de dados.
- Um acumulador (memória) que armazena o valor de energia elétrica consumida desde a primeira conexão à carga, não perdendo este valor mesmo após perder-se a alimentação do módulo. Pode ser zerado com uma ação específica num botão de *clear* quando o módulo está alimentado.
- O μC SD3004 faz a conversão dos sinais lidos com 16 bits, e posteriormente os transmite pela UART em 4 pacotes (um para cada valor V, A, W, Wh), alocando em posições LSB e MSB (a UART recebe pacotes de 8 bits).

A comunicação com o módulo se dá através de comandos AT, comuns a roteadores de internet e outros dispositivos similares, mas esses comandos não foram pesquisados e nem analisados neste trabalho. Os protocolos de comunicação disponíveis para com o módulo estão listados na Tabela 3. Estes são os formatos de comunicação.

[6] Isso para medidores com displays 7 segmentos do produto original, mas na aquisição dos valores via serial não se sabe ainda a range máxima dos valores que serão obtidos.

Tabela 3: Códigos do Protocolo de comunicação e suas funções no PZEM-004T.

Função	Cabeçalho (Head)	Dados Enviados/ Recebidos	Soma (sum)
Tensão	B0 A0	**C0 A8 01 01 00** (envio da requisição do valor de leitura da tensão) 00 E6 02 00 00 (exemplo, responde que o valor da tensão é 230,2V)	1A 88
Corrente	B1 A1	**C0 A8 01 01 00** (requisição leitura corrente) 00 11 20 00 00 (exemplo, responde 17,32A)	1B D2
Potência Ativa	B2 A2	**C0 A8 01 01 00** (requisição potência ativa) 08 98 00 00 00 (exemplo, 2200W)	1C 42
Energia Consumida	B3 A3	**C0 A8 01 01 00** (requisição energia consumida) 01 86 9f 00 00 (exemplo, 99999Wh)	1D C9
Comunicação	B4 A4	**C0 A8 01 01 00** (corresponde ao endereço 192.168.1.1) [7] 00 00 00 00 00 (responde que a conexão foi estabelecida)	1E A4
Alarme[8]	B5 A5	C0 A8 01 01 14 (requisita um valor para definir o alarme de energia) 00 00 00 00 00 (responde que o limite foi definido)	33 A5

Fonte: Peacefair

Cada par alfanumérico é um valor em hexadecimal que corresponde, posição a posição, a um valor em dígitos decimais. Os valores das grandezas lidas são concatenados posição a posição. Ao fazer a concatenação dos hexadecimais, os zeros à esquerda de cada par são desconsiderados, e somente após é realizada a conversão para outra base numérica. O endereço de IP e outros comandos fogem à regra, sendo representados no formato 255.255.255.255.

Exemplos do que foi dito acima podem ser conferidos no Anexo G.

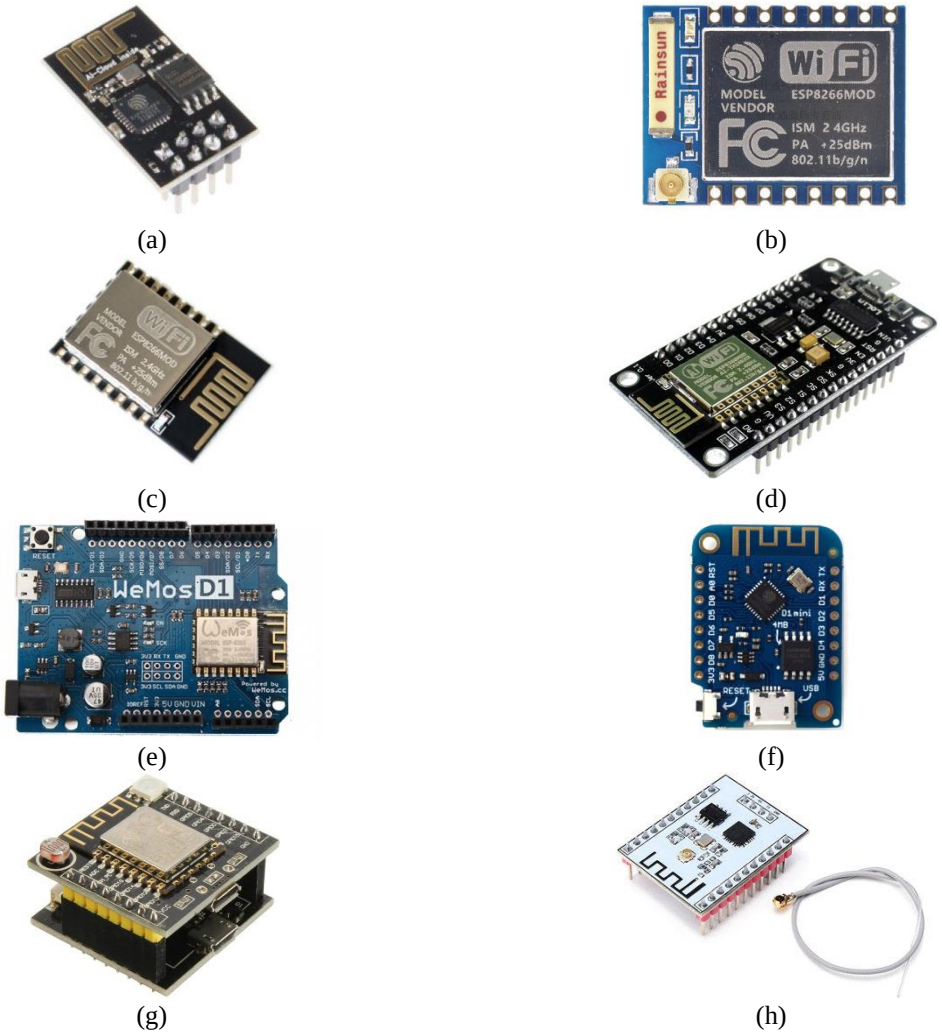
2.7. Placa de Desenvolvimento ESP8266 NodeMCU Lolin e Compatibilidades com a Plataforma Arduino

O microcontrolador ESP8266 é fabricado pela Espressif agrega boa capacidade de memória de armazenamento global e conexão sem fio (WiFi, padrão IEEE 802.11) nativa, sendo também compatível com a maioria dos módulos e códigos da plataforma Arduino.

Trata-se de uma família de microcontroladores ESP8266 que se tornou tendência entre desenvolvedores e entusiastas do "Faça-Você-Mesmo"[9]. Os módulos mais conhecidos e algumas placas derivadas, estão ilustrados na Figura 5 a seguir:

[7] Este endereço inicializa a transmissão de dados entre o módulo PZEM004T e o hardware que estiver requisitando seus dados. Precisa ser enviado a cada vez que o módulo perder alimentação.
[8] Função ainda não testada.
[9] Do inglês "*Do It Yourself"*, popular e comumente abreviado por DIY.

Figura 5 – (a) ESP-01, (b) ESP-07, (c) ESP-12E, (d) NodeMCU Lolin, (e) Wemos D1 *shield* similar Arduino, (f) Wemos D1 Mini (g) ESP-12F, (h) ESP-201.



(a) (b) (c) (d) (e) (f) (g) (h)

Fonte: Google.

Cada uma dessas versões libera um número definido de funções e pinos disponíveis. E por ser um microcontrolador de montagem em superfície, uma placa precisa ser adquirida ou desenvolvida para soldá-lo e acessar seus pinos. Além disso, como todo microcontrolador, ele precisa de um circuito de gravação do firmware, seja adquirido ou montado para ele. Assim, a aquisição de módulos de desenvolvimento são uma vantagem, por propiciarem tanto o circuito de gravação quanto acesso aos pinos do ESP.

2.7.1. NodeMCU Lolin versus Arduino UNO R3

Ao se escolher um microcontrolador para um projeto é preciso saber de antemão algumas informações, entre elas o que se pretende desenvolver e se ele possui todos os recursos necessários.

A Tabela 4 lista as principais características do NodeMCU Lolin e ainda faz um comparativo destas com o Arduino UNO R3.

Tabela 4: Comparativo de Consumo de Alguns Dispositivos Portáteis.

Característica	Módulo	
	NodeMCU Lolin	**UNO R3**
Nível de Tensão	3,3 VCC	5,0 VCC
Entradas e Saídas Digitais (E/S)	17[10]	14
Entradas e Saídas Analógicas	1 (*in*) de 10 bits e 4 (*out*)	6, de 10 bits
Memória Flash	1 – 4 MB	32 kB
Memória RAM	20 kB	2 kB
Memória EEPROM	-	1 kB
Clock Principal	80 – 160 MHz	16 MHz
Alimentação do Módulo	5 -	7 – 12 Vcc
Corrente máxima E/S	12 mA	40 mA[11]
CPU	Tensilica l106 32 bits	AVR 8 bits
Wi-Fi nativo	Sim	Não
Linguagem Programação	LUA[12]	Processing[13]
Barramentos	UART0, UART1,SPI (master/ slave), HSPI (slave), I2C, I2S, IR.	UART0, UART1, SPI, I2C.

Fonte: *Datasheets* das fabricantes Espressif e Atmel.

Destaca-se nessa tabela a questão da linguagem de programação, que na verdade são como camadas de interpretação e não a forma nativa de acesso aos recursos do microcontrolador. Felizmente, ambos os módulos respondem à Interface de Desenvolvimento Integrado (ou do inglês *IDE - Integrated Development Environment*) muito popular aos usuários e iniciantes do Arduino.

Era intenção também fazer outros comparativos com outras placas com a funções Wi-Fi nativas ou adaptadas (como algumas versões de placas Arduino, Intel, etc.), mas não se encontrou muitas informações (condensadas e referenciadas) sobre elas que permitissem comparativos úteis a este projeto.

Usuários de placas Arduino também podem estranhar o modo de usar as portas e pinos de um ESP. A exemplo, o NodeMCU Lolin possui 30 pinos, entre alimentação e outras funções auxiliares. Mas é preciso notar que nem todos os pinos ou GPIOs são funcionais a todas as aplicações. Apenas 9 deles podem ser considerados pinos E/S clássicos, mas ainda

[10] Nesta versão apenas 9 GPIOs (D0 a D8), além do pino A0, estão livres, as demais já são ocupadas ou dedicadas a outras funções.
[11] O máximo suportado somando o consumo de todas as portas é 200 mA.
[12] Referencial completo sobre a linguagem LUA: https://www.lua.org/portugues.html
[13] Um pouco sobre a linguagem *Processing* aqui: http://joaofaraco.com.br/arte_design/introducao-ao-processing/

assim alguns possuem limitações de uso por também serem dedicados a outras funções. É preciso ter atenção a isso tanto ao se escolher a plataforma quando à criação do código.

2.7.2. *A Questão do Grau de Disponibilidade do Serviço*

Uma das questões envolvendo projetos embarcados ou de aplicação IoT é a questão da autonomia da alimentação, pois os módulos precisam de energia para funcionarem. Sistemas de fornecimento ininterrupto de energia (ou do inglês *UPS - Uninterruptible Power Supply*) são empregados para garantir o grau de disponibilidade do serviço.

Felizmente os módulos são muito econômicos, mas ainda assim possuem um consumo que limitará seu tempo de uso, especialmente ao agregar outros módulos ou durante a comunicação de dados.

Para um comparativo, levando em consideração apenas o módulo ESP envolvido no projeto, a Tabela 5 faz um comparativo do consumo de vários dispositivos conhecidos normalmente empregados em *IoT*.

Tabela 5: Comparativo de Consumo de Alguns Dispositivos Portáteis.

Módulo	**Consumo por Modo Operação (mW)**			
	Sleep	**Sem comunicação**	**Recebendo**	**Transmitindo**
Telos Motes	0,015	3	41	41 (0 dBm)
BLE nRF51822	0,0018	7	39	31,4 (0 dBm)
ESP8266	0,033	49,5	165 184,8	561 (17 dBm) 462 (15 dBm) 396 (13 dBm)
Arduino Nano	0,115	75	-	-
Raspberry Pi	-	1150 1500 1650	-	-
Smartphone	150	500	750	1000
Notebook	500	14000	14000	14000

Fonte: Internet das Coisas com ESP8266, Arduino e Raspberry Pi, p. 58.

Observa-se que mesmo comparado com uma das menores versões do Arduino, o ESP ainda leva vantagem com baixo consumo. Novamente, não foram encontradas referências sobre consumo de energia de outros módulos voltados para IoT (conexão Wi-Fi nativa), apenas das capacidades de corrente de portas, quem são muito parecidas entre os módulos pesquisados.

Algumas alternativas para autonomia de operação dos módulos é o uso de baterias recarregáveis e energia fotovoltaica.

2.8. Computação em Nuvem e o Protocolo MQTT

Segundo (OLIVEIRA, 2017, p. 75), computação em nuvem[14] é essencial aos projetos *IoT* por garantir confiabilidade a baixo custo, permitindo dispositivos em ambientes hostis, sem grande confiabilidade, desde que as informações sejam periodicamente armazenadas em nuvem. Assim, dispensando servidores e usufruindo de canais de comunicação de baixa confiabilidade. É algo que na prática seria como terceirizar servidores, como ocorreu desde os primórdios da Internet com a *web* e o *e-mail* (Google, Microsoft, etc.).

No cenário *IoT* ter dispositivos conectados à nuvem produz maior flexibilidade e escalabilidade. Além de disponibilizar um ambiente distribuído que pode ser programado em diversas linguagens e hospedar-se em uma infinidade de servidores.

Os dispositivos precisam desse recurso para enviar e receber informações, comandos. Nesse interim, outro termo que desponta é a "Web das Coisas", onde pesquisadores e projetistas tem agido para o futuro da web. O protocolo já citado LoRaWAN, por exemplo, é um desses esforços de mover as coisas para uma rede própria, e não congestionar as redes já existentes.

Neste trabalho não serão abordados conceitos de Web Services, Protocolos de Comunicação e Bancos de Dados, entre tantos outros assuntos relacionados ao tema, por não terem sido aplicados ao projeto. Mas conhecê-los é importante ao profissional que deseje se dedicar ao assunto.

2.8.1. Protocolo MQTT

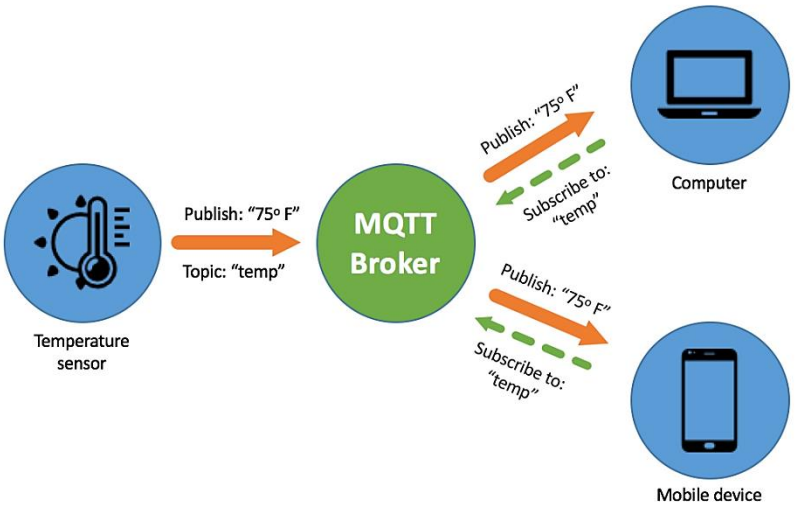
De modo sucinto, é um protocolo "máquina a máquina" (M2M). Conforme explicado por (JAVED, 2017, p. 58), ele segue o modelo de "publicar-assinar" (*publish-subscribe*, no termo em inglês), fazendo uso de um servidor (conhecido como *broker*, do inglês) para que um publicador (*publisher)* envie dados e um assinante (*subscriber*) os receba. Ambos não são conhecidos um do outro, eles apenas conectam-se ao *broker* (comunicação assíncrona). O servidor então avisa a todos os assinantes sobre a publicação de novos dados relevantes, num conceito denominado de "tópicos", que é semelhante a um *feed* de notícias.

---

[14] A computação em nuvem (ou do inglês *Cloud Cumputing*) "é o fornecimento de serviços de computação – servidores, armazenamento, bancos de dados, rede, software, análise e muito mais – pela Internet ("a nuvem"). Fonte: https://azure.microsoft.com/pt-br/overview/what-is-cloud-computing/

Publicadores e assinantes podem ser, mas não se limitar, a sensores, máquinas, aplicativos móveis. A Figura 6 diagrama a simplicidade do processo:

Figura 6 – Diagrama de conexões de dados e medição do PZEM004T.



Fonte: Leor Brenman, 2018[15].

Um mesmo dispositivo pode ser tanto publicador como assinante ao mesmo tempo. Um exemplo seria um aplicativo de celular que recebe informações e com base nelas o usuário pode agir enviando algum comando pelo mesmo aplicativo.

Em geral, servidores MQTT se utilizam da Internet para conexão. E como ocorre com páginas web, existem diversos servidores livres ou pagos, sendo que para usufruir de recursos extras como DNS, banco de dados, web, entre outros, de forma exclusiva, isolada e com camadas extras de segurança apenas nos serviços pagos.

Existe uma gama enorme de servidores livres, inclusive dedicados a ambiente acadêmico, sendo que muitos se aproveitam do ambiente Linux. Alguns servidores possuem interfaces *web* configuráveis ou não. Alguns possuem também aplicativos móveis próprios, outros apenas aplicativos, e outros ainda podem ser usados com aplicativos de terceiros. As opções de recursos são variadas, mas neste trabalho não se teve intenção de verificar a grande maioria deles e seus respectivos servidores.

Como exemplo de servidores pagos tem-se: Microsoft Azure, Google Cloud Plataform, IBM Watson IoT Plataform, Amazon Web Services IoT. Do lado dos servidores livres o mais recorrente foi o Mosquitto[16], da Eclipse Foundation, muito citado em projetos

[15] Artigo API Builder and MQTT for IoT: https://www.appcelerator.com/blog/2018/03/api-builder-and-mqtt-for-iot-part-1/
[16] Mosquitto: https://mosquitto.org/

colaborativos e com vasta documentação. Ainda assim, todos estes servidores requerem conhecimentos mais abrangentes e relativamente avançados, não cabendo aqui sua abordagem.

Felizmente, para facilitar o acesso e uso, algumas Interfaces de Programação de Aplicativos[17] (do inglês *API – Application Programming Interface*) amigáveis foram desenvolvidas por outras iniciativas para os servidores MQTT.

Além disso, plataformas como Arduino e ESP, por exemplo, possuem bibliotecas desenvolvidas e distribuídas especialmente para configurar e trabalhar com estas APIs.

2.8.2. Gerenciador MQTT Blynk

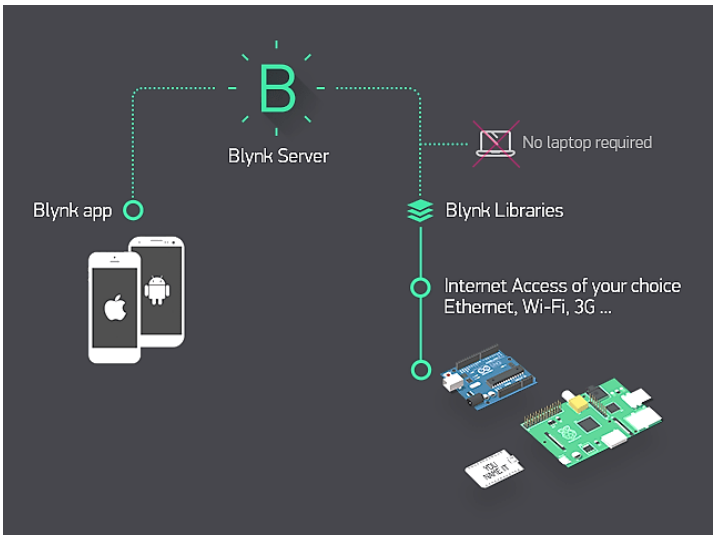
Segundo o site do desenvolvedor[18], o Blynk é uma plataforma de aplicativos que rodam em sistemas iOS® e Android® para uso com Arduino, Raspberry Pi e outros.

O dispositivo precisa apenas estar conectado à Internet - seja por Arduino, ESP8266, ou Raspberry Pi, com Wi-Fi ou Ethernet - que o Blynk poderá então ser usado para um projeto IoT.

Consiste de um painel digital onde é possível criar uma interface gráfica para o projeto arrastando ou soltando *widgets*[19]. Cada *widget* custa um valor diferente de 'créditos' chamados 'energias'. Mais créditos podem ser comprados.

Na Figura 7 consta o resumo da arquitetura do Blynk.

Figura 7 – Arquitetura por trás do Blynk.



[17] API: https://canaltech.com.br/software/o-que-e-api/
[18] Blynk: https://www.blynk.cc/
[19] O termo *widget* refere-se a um componente que pode ser utilizado em computadores, celulares, *tablets* e outros aparelhos para simplificar o acesso a outro programa ou sistema. Eles geralmente contêm janelas, botões, ícones, menus, barras de rolagem e outras funcionalidades. Fonte: https://canaltech.com.br/produtos/O-que-e-widget/

Fonte: Blynk.

Pontos positivos:

- Relativamente de simples configuração, e não se limitando a alguma placa ou módulo[20] específico.
- Ao trocar um *widget* por outro, os valores de ‘energia’ são estornados.

Limitações:

- Possui apenas 2000 créditos de energias.
- Não possui uma interface web para análise de dados ou envio de comandos.

A Figura 8 mostra exemplo de interface para celular:

Figura 8 – Aparência exemplo de interface configurada com *widgets* do Blynk.

Fonte: Blynk.

2.8.3. Gerenciador CloudMQTT

O Blynk é funcional, mas não possui uma interface web para visualização das mensagens trocadas entre publicadores e assinantes, o que limita seu uso a celulares que permitam usar esse aplicativo. Também não é totalmente livre, pois além de não oferecer acesso aos seus servidores tampouco é gratuito após ultrapassar um limite de funções ou recursos. A criação de um link de compartilhamento de interface também gera um ‘custo de

[20] O termo usado foi *shield* (escudo), muito comum entre usuários de Arduino e similares.

energia', mas ao contrário dos *widgets* este recurso do link não é reembolsável depois de usado.

O CloudMQTT[21] é gerenciado por servidores Mosquitto em nuvem. O Mosquitto implementa o protocolo MQTT. O CloudMQTT permite que você se concentre no aplicativo, em vez de gastar tempo no escalonamento do broker ou na atualização da plataforma. É um serviço que pode ser assinado, mas pode ser usado gratuitamente.

Limitações:

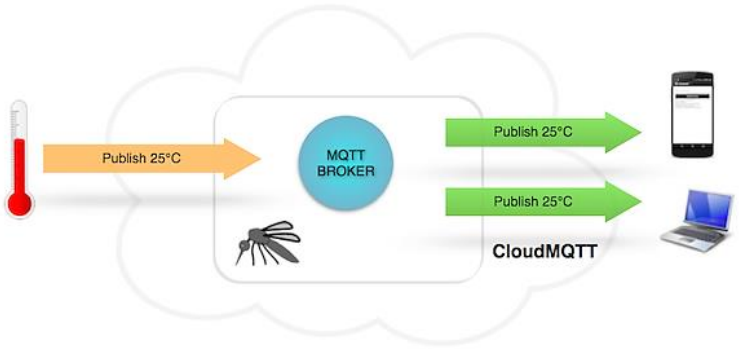
- Não possuir uma interface gráfica web (do tipo *widgets*), respondendo apenas com textos, mas já é um bom começo para conferir tráfego e comunicação.
- Na versão gratuita o número de conexões é limitado a cinco usuários, assim como a largura de banda de 10 Kbit/s.

Pontos positivos:

- Possui uma conexão *WebSoquet*[22] que é usada, como citado, para comunicação via textos na página do servidor.
- Possui aplicativos móveis (como o *MQTTDash* e *MQTT Dashboard*, ambos testados durante este trabalho e semelhantes à interface Blynk) que são compatíveis.
- Compatível com plataformas Arduino e ESP, entre outras.

Exemplos de conexão e interface web do CloudMQTT, visto nas Figuras 9 e 10:

Figura 9 – Exemplo de conexões do CloudMQTT.

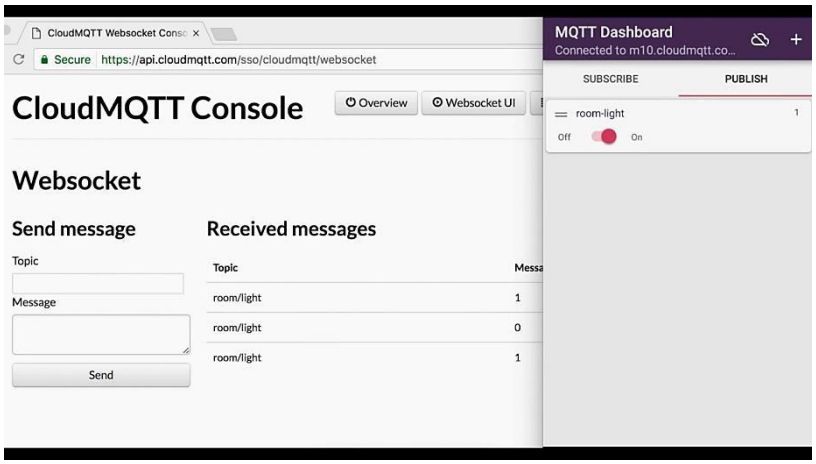


Fonte: CloudMQTT.

[21] CloudMQTT: https://www.cloudmqtt.com/

[22] A especificação *WebSocket* define uma API que estabelece conexões de "soquete" entre um navegador da web e um servidor. Em outras palavras, há uma conexão persistente entre o cliente e o servidor e ambas as partes podem começar a enviar dados a qualquer momento. Fonte: https://www.html5rocks.com/pt/tutorials/websockets/basics/

Figura 10 – Exemplo de aparência das Interfaces web e aplicativo móvel do CloudMQTT.



Fonte: CloudMQTT.

2.8.4. Gerenciador MQTT Cayenne

O Cayenne[23], da empresa myDevices[24], é outra API encontrada e avaliada para este trabalho. É um meio termo entre Blynk e CloudMQTT, associando o melhor de ambos. Eis alguns pontos positivos:

- Possui interface web gráfica, configurável conforme o projeto.
- Possuem um aplicativo configurável próprios, de fácil configuração.
- Compatível com plataformas Arduino, ESP, Raspbery Pi e LoRa.

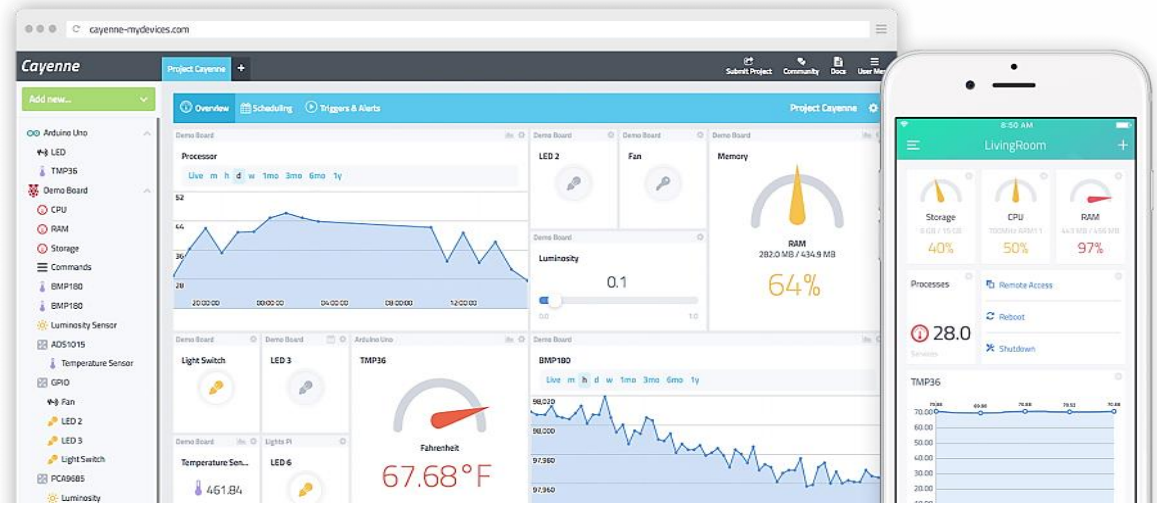
Limitações:

- Também é um serviço pago, mas o que parece a empresa oferece soluções personalizadas para clientes dos mais variados segmentos (e cobra por isso).

Um exemplo de interface com Cayenne por ser visualizado na Figura 11:

[23] Cayenne: https://mydevices.com/cayenne/features/api/
[24] myDevices: https://mydevices.com/

Figura 11 – Interface web e aplicativo móvel customizados do Cayenne, da myDevices.



Fonte: myDevices.

2.9. Módulos Eletrônicos Utilizados neste Projeto

Alguns módulos eletrônicos extras foram escolhidos para integrar este trabalho e permitir que os objetivos propostos fossem atingidos. Detalhamento sobre eles será dado no capítulo apropriado, pois em geral constam de peças não incomuns no meio eletrônico.

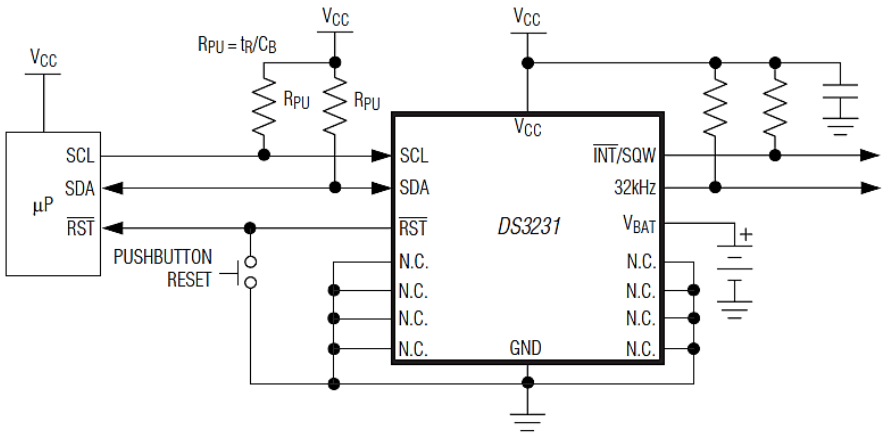
2.9.1. Relógio de Tempo Real (RTC) e o Cartão de Memória

O dispositivo que registra o tempo de eventos é essencial num projeto de datalogger, por não fazer sentido ter os dados lidos, mas não se saber quando ocorreram.

Um relógio de tempo real (do inglês RTC – Real Time Clock) consta basicamente de um sistema que permite não apenas ajustar uma referência de tempo como também, a partir deste ajuste, manter este ajuste sincronizado com os demais sistemas — mantendo datas e horários, por exemplo, corretamente. A versão deste trabalho precisava ser digital, possuindo características de manter os valores ajustados independente da alimentação principal dos módulos ou da presença de dados a serem medidos.

O módulo RTC escolhido foi um baseado no CI DS3231, cujas informações básicas podem ser vistas na Figura 12 e na Tabela 6:

Figura 12 – Diagrama esquemático de exemplo de aplicação do RTC DS3231.



Fonte: Datasheet DS3231.

Tabela 6: Características Principais do Circuito Integrado RTC DS3231.

Característica	Valores
Tensão de Operação	3,3 Vcc (2,3 a 5,5 V)
Temperatura de Operação	0º a 70ºC
Tensão da Bateria	3,3 Vcc (2,3 a 5,5 V)
Corrente de Consumo	200 µA (110 a 650 µA)
Sensor de Temperatura do CI	± 3ºC de precisão

Fonte: Datasheet DS3231.

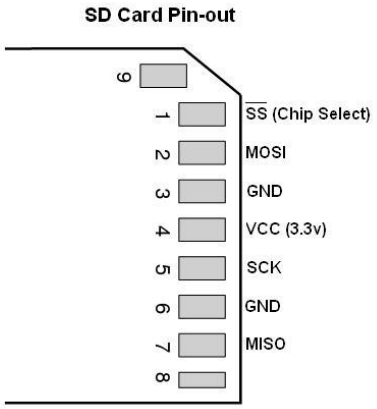
Tão importante quanto quando se precisam armazenar esses valores num local não volátil ou instável é a necessidade de uma mídia física que permita sua leitura posterior. Tal solução poderia ser tanto analógica (impressa, por exemplo) como digital (num cartão de memória, a um custo bem menor).

SD[25], miniSD, microSD, xD, Memory Stick e MMC são apenas alguns dos tipos de cartão de memória disponíveis no mercado. Embora executem bem o mesmo propósito, seja salvar músicas, fotos, vídeos, etc., a confusão sobre os diversos tipos é comum. Para este trabalho não é essencial discorrer sobre os diversos tipos.

O diagrama de pinagem de um cartão de memória SD pode ser observado na Figura 13:

[25] De *Secure Digital*, estando disponível em pelo menos três tamanhos dimensionais distintos.

Figura 13 – Diagrama esquemático de exemplo de aplicação do RTC DS3231.



Fonte: Google.

2.9.2. Interface Visual de Dados com Interface Digital

Neste trabalho foi inserido um visor de cristal líquido (do inglês *LCD – Liquid Cristal Display*) cuja única finalidade é conferir os valores de leitura e status do sistema sem precisar a recorrer a uma conexão com Internet (que pode estar indisponível).

É um item não crucial, e poderia até ser excluído ao final, mas permite uma redundância de testes visuais que pode ser útil em casos onde se tem dúvida do funcionamento do sistema.

2.9.3. Fonte de Alimentação

O protótipo faz medições de sinais de grandezas elétricas, mas para isso ser possível precisa ser alimentado com a energia necessária para o seu funcionamento.

A vantagem do baixo consumo de energia se perde um pouco com o acréscimo dos módulos de display LCD e do Leitor/Gravador de cartões de memória, que são os itens de maior consumo. O tráfego contínuo de sinais Wi-Fi em curtos espaços de tempo também contribui para menor autonomia.

A melhor solução para um caso como este é utilizar baterias recarregáveis em paralelo com uma fonte principal de energia (como a própria sendo medida), atuando em sua falta. Mas para os propósitos deste projeto, apenas a própria rede elétrica disponível será utilizada.

Os níveis corretos de tensão e corrente de cada módulo precisam ser supridos, pois há necessidade tanto de 5,0 como de 3,3 volts, o que requer reguladores de tensão. A tensão da rede local também é elevada (220 volts) e ao mesmo tempo do tipo alternada, precisando ser convertida.

A confiabilidade da alimentação dos módulos é crucial ao bom funcionamento e segurança do projeto.

3. DESENVOLVIMENTO DO PROTÓTIPO

Este capítulo aborda todo desenvolvimento prático e resultados deste trabalho. Procura dar detalhamento do processo, problemas encontrados e soluções aplicadas.

DEFINIÇÕES: Mesmo que módulos de desenvolvimento ESP8266-12F e ESP-201 tenham sido utilizados em alguns testes de código e compatibilidade, para fins de clareza, todos os resultados citados aqui estão relacionados ao NodeMCU Lolin, portanto, passam a ser referidos apenas como ESP no restante deste trabalho. Também será apenas utilizado o termo PZEM ao se referir ao módulo que faz as leituras de tensão e corrente e uC para microcontroladores.

3.1. Escolha pelo Módulo de Medição de Energia PZEM004T

Esta etapa do projeto é fundamental, obviamente, pois o que se precisa é justamente medir algo — o que fazer depois e consequência do sucesso nessa empreitada.

As primeiras tentativas foram de usar sensores simples e fazer todo tratamento do sinal via hardware e software. Por questões de limite de tempo e diversos contratempos tomou-se conhecimento de módulos prontos que resolviam problemas de precisão e confiabilidade da leitura das grandezas necessárias a este trabalho.

O ideal para um projeto de sensores modulares seria desenvolver um com base em sensores analógicos (sensores, transdutores, circuitos eletrônicos) ou digitais (com a aquisição de um SoC por menos que U$ 5[26]). O que se percebeu foi que isso, por si só, demandaria um trabalho único de pesquisa e desenvolvimento, detalhando demais o que seria necessário para um *datalogger*.

Por este e outros motivos, saber dos módulos da Peacefair durante a pesquisa propiciou motivação suficiente para testar como solução ao projeto. Mesmo não podendo ser considerado um sistema de Código Aberto, seu custo de aquisição e possíveis facilidades foram um incentivo ao seu uso.

Deste ponto em diante, ao se referir ao módulo utilizado se usará apenas a referência a PZEM, módulo de medição ou módulo (subentendido).

---

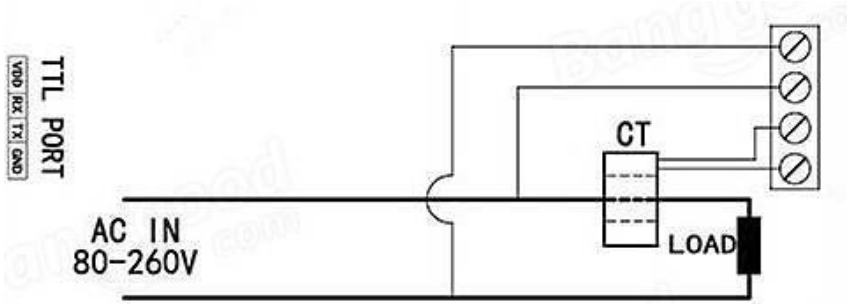
[26] Como este http://www.st.com/en/data-converters/stpm01.html#samplebuy-scroll ou este https://www.cirrus.com/products/cs5463/ exemplos.

3.1.1. Diagrama de Ligações à Rede Elétrica

Por não ser de código aberto, a documentação e detalhamento que poderiam ser úteis para solução de diversos problemas ou até de novas bibliotecas fica comprometida. Ainda que um sistema funcional possa ser montado, algumas limitações ou problemas podem ser de difícil solução.

O modelo de módulo adquirido não possuía uma caixa e nem os displays LED 7 segmentos, consistindo apenas do módulo com seu TC. Conexão do módulo é detalhada na Figura 14:

Figura 14 – Diagrama de conexões para medição do PZEM004T e também pinos da porta TTL.



Fonte: Peacefair.

O sensor de corrente trata-se de um transformador de corrente (indicado por CT). Os pinos de conexão de dados e alimentação do módulo situam-se no lado oposto da placa, com isolação por opto-acopladores. Todas as conexões são rápidas e sem complicações, com marcações claras através da serigrafia da placa.

É de vital importância que tanto o CT quando os fios fase e neutro não sejam conectados aos terminais errados do PZEM. Portanto, deve se seguir exatamente a ordem mostrada. Considere a vista da Figura 14 com igual a da vista superior da Figura 4(b).

Também o manuseio do módulo deve ser realizado com cuidado para evitar curtos-circuitos e choque elétrico, pois todos os pontos de solda estão disponíveis ao contato. Neste trabalho foi sempre usada uma superfície em madeira ou de vidro onde o PZEM foi testado, estando a bancada completamente limpa de resíduos ou outros objetos.

3.2. Preparando a IDE Arduino para Trabalhar com ESP.

Já foi citado que as placas ESP podem ser gravadas via a IDE da plataforma Arduino. Para que isso seja possível, é necessário instalar algumas bibliotecas e fontes externas. O modo de fazer isso está escrito sucintamente no Anexo F.

A numeração que aparece impressa no corpo do NodeMCU Lolin não tem qualquer relação com o número dos terminais do ESP8266 que são declarados dentro do código escrito na IDE. A pinagem equivalente do NodeMCU pode ser conferida no ANEXO D. Onde, por

exemplo, “0” (D0) gravado na placa deve ser declarado no código como “16” (GPIO16) e não “0”.

3.3. Desenvolver com Arduino (UNO R3) ou ESP (NodeMCU Lolin).

Se a correta leitura dos sinais é a primeira etapa a ser resolvida, esta etapa intermediária do trabalho é a que determina o ‘cérebro’ no controle e comando de todo o processo. Assim é necessário determinar qual melhor custo *versus* benefício.

Primando pela agilidade que os prazos requeriam, a opção foi por plataformas que precisavam atender aos seguintes critérios:

- Que a preocupação principal não fosse a polarização correta do uC (que demanda tempo e incorre e em diversos erros de prototipagem);
- A linguagem de programação fosse de simples aprendizado;
- Que houvesse fácil acesso a algum suporte profissional em caso de dúvidas;
- O custo do protótipo não fosse superior ao do “equipamento” a ser medido;
- Fácil disponibilidade de peças e módulos compatíveis, com bibliotecas e exemplos acessíveis.

Além disso, o projeto não se destina à produção em larga escala, então a questão de custo poderia ser sacrificada um pouco sem grandes prejuízos de aprendizado e propósito.

No começo do trabalho a plataforma escolhida foi o Arduino, com a sua placa UNO R3. O projeto do *datalogger* foi concluído (e apresentou-se funcional) nesta etapa, mas a seguir foi solicitado que uma interface independente via Internet fosse incorporada para leitura em tempo real. Então uma decisão precisou ser tomada, sob o seguinte cenário:

- Qual tipo ou ferramenta de interface em tempo real poderia ser usada?
- O que atenderia melhor a um protótipo sem o foco em um aparelho que pudesse ficar no padrão de medição de energia, ou então que fosse portátil?
- Que modificação permitiria ainda para que o projeto não precisasse de grandes intervenções de infraestrutura no local de seu uso?
- Como aproveitar todo o trabalho de pesquisa e desenvolvimento já realizado?
- O Arduino utilizado já estava com boa parte de sua capacidade de armazenamento aproveitada, o que poderia gerar incompatibilidades ao agregar novas funções sem um tratamento de código intenso. Isso serial temporalmente viável?

A solução se apresentou em módulos ESP8266 então disponíveis naquele momento e em sua apregoada capacidade de conexão com a Internet aliada à capacidade de memória e baixo custo.

A questão que agora precisava ser respondida é se os mesmos módulos e bibliotecas disponíveis no Arduino poderiam ser aproveitados no ESP. É o que será abordado a seguir.

O código deste projeto em sua íntegra pode ser lido no Anexo C deste trabalho, e não será transcrito no corpo principal deste. O código também poderá ser baixado no repositório do GITHub[27] onde também dúvidas poderão ser respondidas.

3.3.1. Biblioteca e Comentários do Código Base do PZEM Rodando sobre o ESP

Como percebido, para coletar os dados lidos pelo PZEM é necessário que se envie e se leia linhas de código, o que pode ser bem trabalhoso quando se deseja um desenvolvimento de um sistema com certa rapidez. Além do mais, já foi citado que não há documentação clara nem sobre o módulo e nem sobe o SD3004. Felizmente, para uso na plataforma Arduino, foi criada uma biblioteca que permite acesso e leitura do PZEM (SOKOLOV, 2018), cujos comandos principais serão citados a seguir.

As Figuras 15 e 16 apresentam trechos de código para teste de funcionamento do PZEM na IDE Arduino, comentando cada linha necessária a seguir:

Figura 15 – Código exemplo de teste do PZEM004T, com a biblioteca, variáveis globais e configurações de inicialização.

```
 1 #include <SoftwareSerial.h>   // IDE inferior a 1.6.6
 2 #include <PZEM004T.h>
 3
 4 PZEM004T pzem(10,11);         // IOs do RX, TX
 5 IPAddress ip(192,168,1,1);
 6
 7 Void setup() {
 8 Serial.begin(9600);
 9 Pzem.setAddress(ip);
10 }
```

Fonte: SOKOLOV (código).

A biblioteca <SoftwareSerial.h> é necessária apenas para versões inferiores a 1.6.6 da IDE. A inclusão da biblioteca <PZEM004T.h> permite manipular os dados do PZEM com linhas de código mais simples e inteligíveis. Esta biblioteca não se mostrou completamente compatível com o ESP, e ao contrário do Arduino, poucas foram as portas em que foi possível fazer leituras. Isso é perfeitamente compreensível, pois se tratam de arquiteturas de hardware diferentes.

O comando “IPAddress ip(192,168,1,1)” não tem relação com conexões de internet neste caso, mas é o mesmo usado em bibliotecas de conexão, justamente por que comando de inicialização do PZEM tem este formato. Este é um valor que não deve ser alterado, nem as vírgulas trocadas por pontos. Declarar com o nome “ip” facilita a escrita repetitiva do código,

[27] GITHub: https://github.com/gersonsena/NodeMCULolin_PZEM004t_Meter

mas deve ser mudado caso outras funções de configuração IP sejam testadas (o que não foi realizado neste trabalho).

O comando "PZEM004T pzem(10, 11)" indica em quais pinos do ESP os dados vão ser enviados e recebidos (RX, TX). Neste projeto precisaram ser mudados para as GPIO's 00 e 02 (D3 e D4 respectivamente), pois outros dois pares de pinos possíveis foram ocupados para outros propósitos.

Mostrou-se necessária a inclusão de um laço *while* (ver o código no Anexo C) para forçar a conexão do módulo tanto no Arduino quanto como no PZEM. Quando a conexão não era estabelecida na primeira tentativa eram necessários diversos *resets* físicos na placa até que fosse possível comunicar. O laço foi inserido no *setup* por ser necessária apenas uma requisição com sucesso enquanto o PZEM permanecesse alimentado[28].

Também é importante frisar que o módulo PZEM apenas consegue se conectar (inicializar comunicação) se houver sinal de tensão em sua entrada (pinos AC da Figura 4(b)), independente dele estar alimentado. Após realizada a conexão, a tensão pode cair (falta) que o módulo continuará a comunicação sem necessidade de *resetar* o sistema.

O *baudrate* usado foi o de 115200, diferente dos 9600 mostrados.

Figura 16 – Segunda parte do código agora mostrando o *loop* de execução do programa.

```
12 Void loop();{
13 float v = pzem.voltage(ip;
14 if (v < 0.0) v = 0.0;
15 Serial.print(v); Serial.print("V; ");
16
17 float i = pzem.current(ip);
18 if (i >= 0.0) {Serial.print(i); Serial.print("A; ");}
19
20 float p = pzem.power(ip);
21 if (p >= 0.0) {Serial.print(p); Serial.print("W; ");}
22
23 float e = pzem.energy(ip);
24 if (e >= 0.0) {Serial.print(e); Serial.print("Wh; ");}
25
26 Serial.println();
27
28 Delay(1000);
29 }
```

Fonte: SOKOLOV (código).

As variáveis foram declaradas como tipo *float*, mas apenas as de tensão (v) e corrente (i) apresentam casas decimais.

Após estabelecida a comunicação, se o sinal de tensão cair, e o PZEM continuar alimentado, os valores as variáveis assumem todos um valor pré-determinado que ao ser

[28] Refere-se a alimentar a placa do PZEM com os 5Vcc que ele precisa para executar suas funções, e não deve ser confundida com a tensão a ser medida da carga em teste.

convertido em decimal resultaria “-1”. Que indica que não há tensão da rede elétrica na entrada de AC.

A variável de energia consumida (e) não é zerada mesmo que falte energia e a alimentação do PZEM seja interrompida. Para que este valor se torne zero (0 kWh) é preciso, enquanto o PZEM estiver alimentado, segurar o botão de *clear* (clr) da placa por cinco (5) segundos ou pouco mais, então soltar e tornar a pressionar rapidamente. Repetir até funcionar.

Um intervalo pode ser necessário após completar cada ciclo de leitura, para dar condições do ESP (ou Arduino) ler os dados corretamente.

Incompatibilidades da biblioteca <PZEM004T.h> com a arquitetura do ESP provocaram diversos erros de interpretação dos bits lidos. Os valores estavam nos pacotes, mas por algum motivo o ESP não os conseguia converter corretamente, retornando “-1” mesmo quando se sabia que o valor havia sido corretamente lido pelo módulo PZEM.

3.3.2. Conexões Físicas do PZEM com o ESP

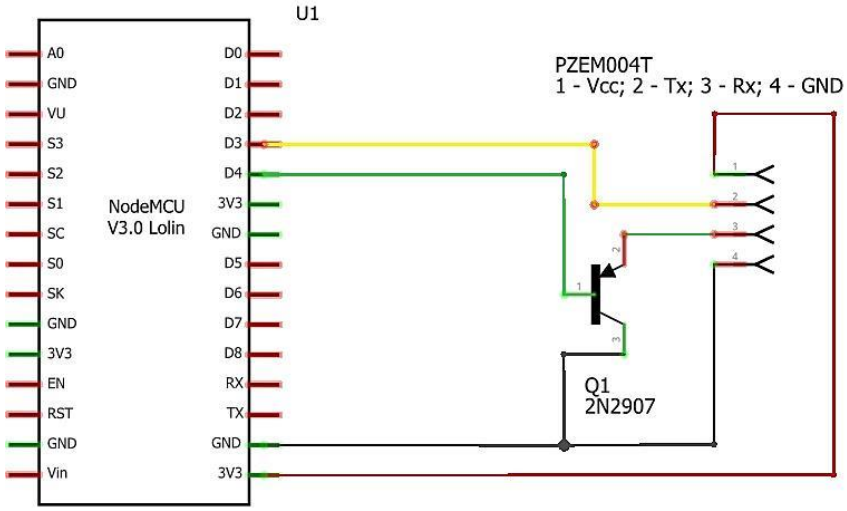
Como já se deve ter percebido, o módulo PZEM é alimentado com 5,0 Vcc, tendo uma comunicação nível TTL. Já o ESP possui entradas e saída em nível 3,3 Vcc (CMOS).

Também apenas 3 pares de GPIOs do ESP se mostraram funcionais em estabelecer a comunicação com o PZEM (sendo um deles os respectivos Rx-Tx do ESP, também aqui usados para debug via Monitor Serial da IDE).

Assim, ao invés de uma ligação direta, foi necessário readequar o sinal para que a comunicação entre PZEM e ESP fosse possível. Haveria alguns modos alternativos de equalizar estes sinais, mas o que foi utilizado e funcionou foi de fazer a alimentação do PZEM (5,0V) em separado e readequação apenas dos sinais de comunicação (RX, TX).

A Figura 17 mostra o esquema elétrico de fiação:

Figura 17 – Diagrama de elétrico de conexão da porta TTL do PZEM ao ESP.



Fonte: Autor, editado no Fritzing.

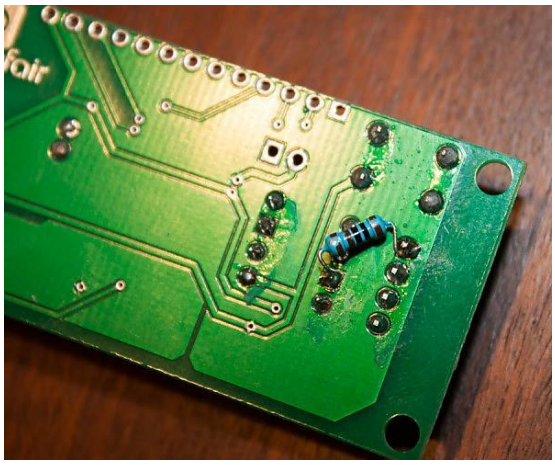
Foi realizada uma alteração física na saída da conexão da porta TTL do PZEM com a inserção de um resistor de 1 kΩ de 0,25 W em paralelo com o resistor SMD (também de 1 kΩ) do opto acoplador da saída RX.

Tal inserção foi para conseguir um divisor de tensão e deixar o sinal TTL mais próximo de 3,3V.

...

A alteração pode ser visualizada na Figura 18:

Figura 18 – (a) Detalhamento da inserção do resistor de 1k no PZEM e (b) Diagrama elétrico da alteração.



(a) ... (b)

Fonte: https://mysku.ru/blog/china-stores/43331.html, autor.

Após estas modificações as portas GPIOs D1, D2, D3. D4 do ESP tiveram resultados positivos em reconhecer e ler os dados do PZEM.

O diagrama de fiação usado neste trabalho pode ser visto no Anexo A.

3.3.3. Tratamento de Leituras Erráticas

Como já citado, toda a biblioteca que propicia a conexão dos hardwares dos módulos foi desenvolvida com base na arquitetura Arduino para o PZEM. Para readequar qualquer problema que pudesse vir a existir, a alteração ou criação de uma nova biblioteca seria o ideal, mas isto foge desta proposta. Ainda assim, para conexões com ESP várias publicações web não citaram encontrar problemas em seus projetos.

O que ocorreu neste trabalho é que em ciclos aleatórios, após algumas leituras, valores estranhos e errados eram percebidos. Como já explicado, quando um valor de sinal está ausente (ou é lido incorretamente) o PZEM envia como resposta o valor "-1". Isso também ocorre sempre que não houver sinal de tensão na entrada. Mas mesmo na presença de sinais em alguns momentos esse valor era devolvido.

O que gerou mais estranheza foi de que o normal era uma linha inteira ser preenchida com "-1", o que não ocorria, indicando provável corrupção dos pacotes lidos.

Depois de tentativas de alguns tratamentos via código como média móvel ou laços condicionais de valores, o que resolveu o problema foi um laço comparativo de valores, pegando apenas o valor maior de cada amostra. Considerando:

- Para o propósito deste trabalho, não era fundamental que cada leitura fosse aproveitada. Bastavam apenas alguns valores amostrados em intervalos de alguns segundos (o que é uma eternidade para microcontroladores), logo algumas leituras poderiam ser desconsideradas.
- Levando em conta também que numa ocorrência de falta de frações de segundo não eram cruciais, estabeleceu-se que seria considerada uma falta apenas se três sequências de leituras indicassem isso (e não apenas uma).

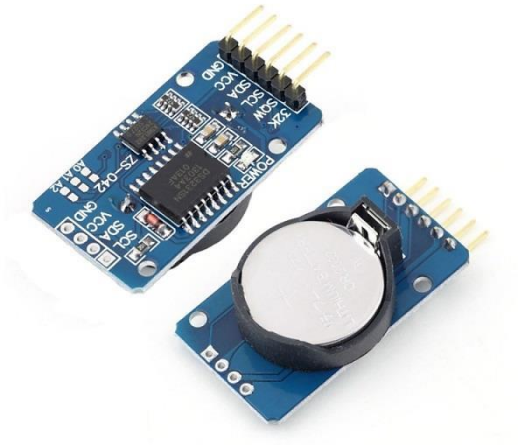
Assim, a cada três ciclos de extração dos dados do PZEM, estas seriam comparadas uma a uma para mostrar apenas o maior valor entre elas.

Após testes com resultados positivos, a função precisou ser aplicada a cada variável enviada do PZEM ao ESP, pois o problema não era nos dados do PZEM, mas sim no modo como o ESP conseguia ler e interpretar estes.

3.3.4. Usando módulo RTC.

O módulo de RTC utilizado neste trabalho é mostrado na Figura 19:

Figura 19 – Módulo RTC DS3231.



Fonte: Google.

O módulo DS3231 usa uma bateria CR2032 (que pode ser recarregável) e é alimentado em 3,3 Vcc, mas suporta bem até 5,5 Vcc.

Não existe muita documentação em português sobre o DS3231 citando suas características, como:

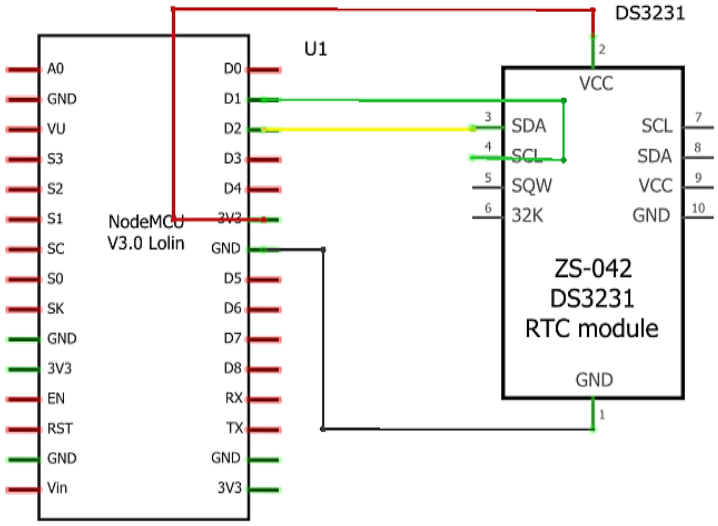
- Definição de alarmes e medição da temperatura.
- O módulo mostrado disponibiliza ainda uma segunda interface I2C para ligar um dispositivo em cascata, como um LCD, por exemplo.
- Também possui uma memória EEPROM (AT24C32) de 32K, útil para armazenar a configuração do relógio e como *datalogger***.** O endereço da memória pode ser configurado através dos jumpers A0, A1 e A2.
- O pino SQW é útil para o controle de alarmes por interrupções.
- O pino 32k raramente é utilizado, apenas se quiser obter os pulsos do oscilador de cristal (para escovadores de bits).

Essas funcionalidades não foram exploradas aqui.

O módulo tem comunicação através de um barramento I2C. Por isso as bibliotecas utilizadas foram a <Wire.h> e <RTCLib.h>, a primeira para comunicação I2C e a segunda para leitura e gravação das informações essenciais do RTC. Caso seja necessário fazer uso de outras funções do módulo, vão ser necessárias, provavelmente, outras bibliotecas.

O diagrama de ligações ao ESP é mostrado na Figura 20:

Figura 20 – Diagrama elétrico do módulo RTC DS3231 ao ESP.



Fonte: Autor, editado no Fritzing.

Note que os dados lidos e escritos correspondem às ligações das GPIOs 05 e 04 (respectivamente D1 e D2).

O diagrama de fiação do RTC pode ser analisado no Anexo A.

3.3.5. Usando módulo SD Card.

O módulo do cartão SD utilizado neste trabalho é mostrado na Figura 21:

Figura 21 – Módulo SD Card.

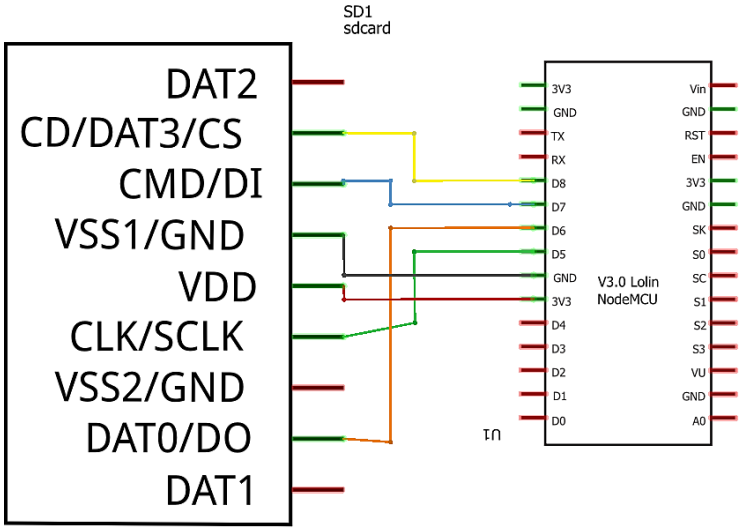
Fonte: Google.

Consta de uma simples placa de circuito impresso com soquete para cartões SD grandes ou adaptadores, regulador de tensão e componentes discretos de polarização. A alimentação desta placa pode ser tanto em 5,0 Vcc quanto em 3,3 Vcc, mas em virtude do consumo de energia durante a gravação do cartão, optou-se por deixar na no barramento de 5,0 Vcc.

O cartão usado durante o trabalho foi uma mídia de 2GB de capacidade formatada no padrão FAT32.

Há diversos modelos de leitoras/gravadoras de cartão, inclusive algumas placas já com RTC acoplado. O diagrama de fiação da ligação ao NodeMCU é o da Figura 22, que segue:

Figura 22 – Diagrama elétrico do módulo SD Card ao ESP.



Fonte: Autor, editado no Fritzing.

Este tipo de módulo em geral faz uso do barramento SPI. A biblioteca utilizada com o SD Card no ESP foi a <SD.h> que é referenciada à biblioteca <SPI.h>, mas esta última não precisou ser declarada, embora estivesse instalada na IDE.

O diagrama de fiação pode ser analisado no Anexo A.

3.3.6. Conexão à Internet Wi-Fi.

Para usar os recursos de um servidor em nuvem é necessário acesso à Internet. A fim de facilitar configurações durante toda a fase de testes foi criada uma rede doméstica com roteador Wi-Fi. Isso permitiu melhor dinâmica durante os diversos deslocamentos com o protótipo.

Infelizmente, por conta das políticas de segurança e arquitetura das redes de internet do campus universitário, um roteador não poderia ser anexado à rede, o que exigiria do projeto conexão direta do ESP com a rede institucional, via MAC ID e IP Dinâmico.

Quanto a este contratempo não haveria qualquer problema, porque o ESP trabalha bem com essas configurações de rede, mas o fato de exigir credenciais institucionais do usuário para estabelecer a conexão bloqueou o acesso das placas à Internet provida pela rede institucional, pois não se descobriu um meio de enviar estes dados diretamente via ESP.

Este problema não teve solução até o momento da redação deste trabalho. Não se encontrou quaisquer referências sobre casos deste tipo e como proceder. E tanto o tempo como os conhecimentos necessários para solucionar isso estão além do escopo deste trabalho.

O roteador disponível foi um Multilaser M150, cuja única configuração necessária foi estabelecer o padrão de IPs, nome e senha da rede sem fio e monitorar as placas conectadas. Na Figura 23 consta o modelo usado.

Figura 23 – Roteador da Multilaser, N150.

Fonte: Google.

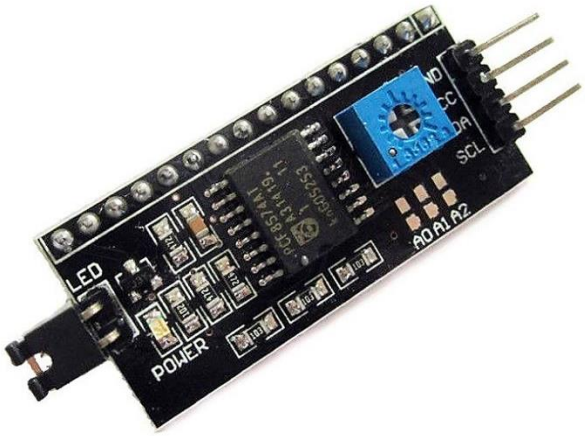
O roteador personalizado deu agilidade ao processo de testes de conectividade e tráfego de dados.

3.3.7. Display LCD Usando Módulo Serial PCF8475A

Dada a pouca disponibilidade de GPIOs livres no ESP a alternativa é a possibilidade de usar barramentos seriais. No caso do display LCD isto então é imprescindível, dado o número de saídas que seriam necessárias. Assim, o barramento I2C também foi usado

Uma das vantagens das filosofias de hardware aberto é a oferta de módulos com soluções prontas. Este é o caso do módulo PCF8574T mostrado na Figura 24, que beneficia o projeto com o uso do barramento I2C:

Figura 24 – Módulo PCF8574T.



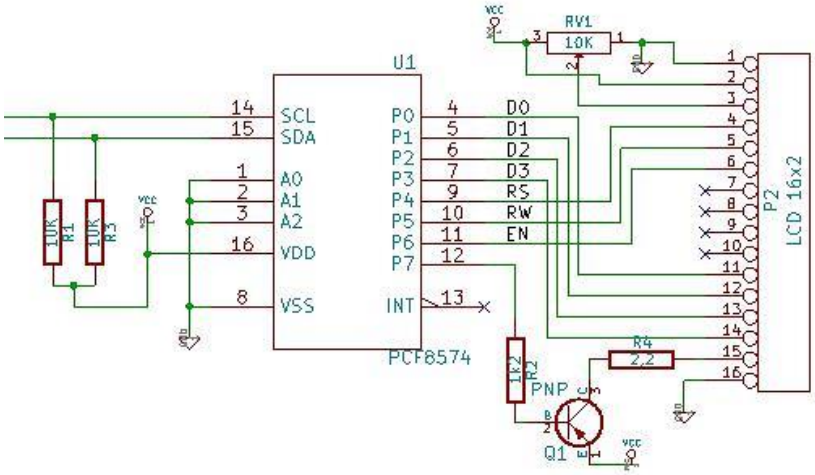
Fonte: Google.

O *jumper* “LED” permite a opção de controlar o *backlight* do LCD via código ou via hardware. Os pontos “A0, A1, A2” permitem alterar o endereço de comunicação do módulo. São possíveis mais de um módulo aplicado no mesmo barramento simplesmente alterando a conexão de algum desses pontos.

Os pinos “SDA" e “SCL” são ligados nas mesmas portas GPIO do módulo RTC já discutido. Para conferir possíveis conflitos de endereços, um código[29] de teste foi rodado permitindo identificar os endereços atribuídos a cada módulo conectado no ESP.

O diagrama esquemático pode ser visualizado na Figura 25:

Figura 25 – Diagrama esquemático do módulo PCF8574T.



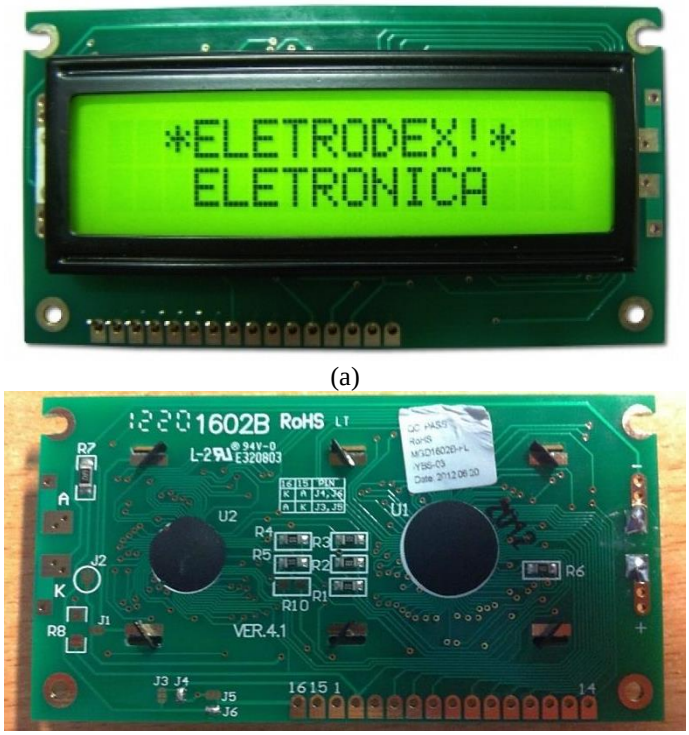
Fonte: Google.

[29] Código e Informações em: https://playground.arduino.cc/Main/I2cScanner.

Observe que na prática apenas 4 pinos de dados do display são usados na comunicação com o módulo.

O módulo display LCD utilizado foi o MDS 1602, mostrado na Figura 26:

Figura 26 – Display LCD MGS 1602B, (a) frontal e (b) traseira.



(a)

(b)

Fonte: Eletrodex e Laboratório de Garagem.

Este módulo também possui 16 pinos, sendo 8 pinos para comunicação de dados e os demais para controle e alimentação do módulo. É preciso atentar que os pinos 16 e 15 deste módulo ficam ao lado do pino 1 e não na sequência que seria esperada.

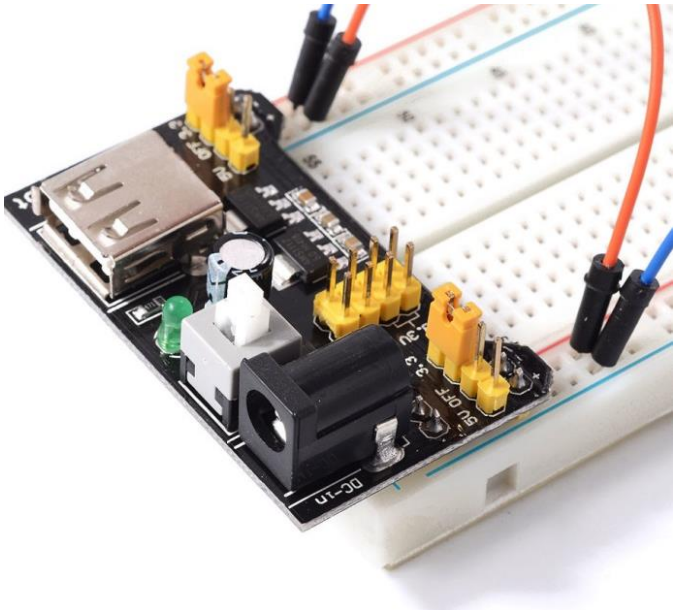
Para o uso deste display juntamente com o módulo PCF8574T foi utilizada a biblioteca <LiquidCrystal_PCF8574.h> no ESP. Outras bibliotecas não foram testadas no ESP até este momento.

3.3.8. Alimentação dos Módulos

Por se tratar de um sistema que deverá operar autonomamente essa é uma questão crucial que precisa ser pensada e projetada em conjunto.

Foi escolhida uma fonte de alimentação com capacidade de 1,0 A, que converte 220 V_{AC} para 15 Vcc (que é mais que suficiente para o projeto). Essa fonte alimenta o módulo da YwRobot Corporation (chinesa) com reguladores de tensão tanto para 5,0 Vcc quanto para 3,3 Vcc através de um plugue P4. Mais detalhes na Figura 27:

Figura 27 – Módulo fonte de alimentação para Protoboard com reguladores de tensão de 5,0Vcc e 3,3Vcc da YwRobot.

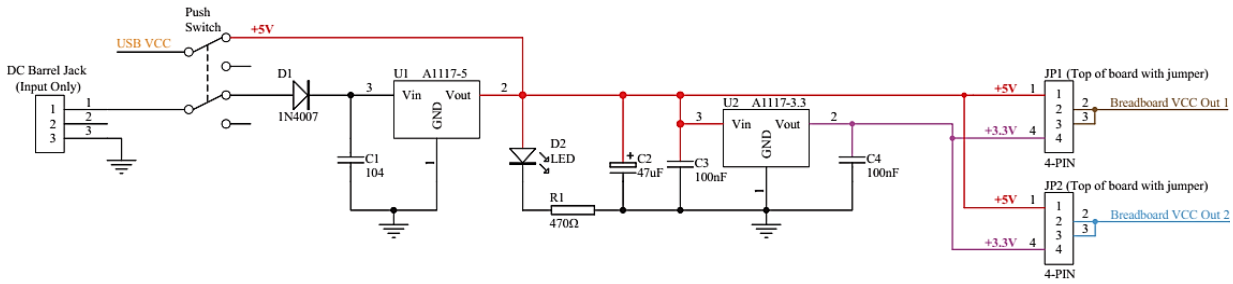


Fonte: YwRobot.

No módulo também há uma porta USB tipo A, um LED indicador de funcionamento, uma chave ON/OFF (também delimita se a alimentação vem pela USB ou pelo Plugue P4), jumpers que selecionam se o barramento da *protoboard* é de 5,0 Vcc ou 3,3 Vcc. Há ainda saídas de 8 pinos para conexão sobre o módulo.

A Figura 28 mostra o diagrama elétrico (a versão completa está no ANEXO B):

Figura 28 – Diagrama elétrico do módulo YwRobot.



Fonte: Addicore.

Conforme descrito nos datasheets dos reguladores, essa fonte tem corrente nominal de 1,0A, mas pode suportar picos de até 1,2ª.

3.4. Configuração do Servidor MQTT no NodeMCU Lolin

A etapa final do projeto é fazer com que todos os dados contidos no ESP sejam enviados a um broker MQTT. Como já citado, havia muitas opções para isto. Neste trabalho testamos os servidores CloudMQTT, Cayenne e Blynk, cujas bibliotecas e principais detalhes serão tratados em sequência.

Cabe lembrar que há diversos códigos de configuração e mapeamento de redes disponíveis para o ESP e o Arduino. Códigos para identificar Endereços, MAC, redes disponíveis, etc., estão disponíveis em fóruns online. Cada vez que uma dessas informações foi necessária para configurar os servidores ou a rede, estes recursos forma utilizados.

3.4.1. Usando o Blynk.

O teste do aplicativo Blynk consistiu no registro de usuário e *download* dos arquivos de configuração para IDE Arduino e aplicativo para o *smartphone* (Android). Logo a seguir, foi realizada a configuração do aplicativo e teste do código gravado no ESP (ver Anexo E).

3.4.2. Cayenne...

Em desenvolvimento...

3.4.3. CloudMQTT...

Em desenvolvimento...

4. RESULTADOS

Nesta seção é possível ver o resultado final do trabalho através de registros visuais e análise dos valores reportados, bem como do comportamento do protótipo durante os períodos de ensaios.

Com base nestas informações é possível abstrair questões como funcionalidade, coerência de leituras, viabilidade do projeto, possíveis aplicações e melhorias, e se os objetivos foram satisfeitos.

4.1. Investimentos Dedicados ao Projeto.

A Tabela 7 lista os valores que foram destinados para execução do protótipo.

Tabela 7: Valores Investidos no Protótipo.

Módulo ou Componente (Local Compra)	Valor (R$)
PZEM004T (China)	24,00
NodeMCU Lolin (Mercado Livre)	29,90
Módulo Alimentação Protoboard (China)	12,10
Módulo RTC (China)	3,90
Módulo SD Card (China)	1,80
Display LCD MGS1602 (China)	4,90
Módulo PCF8574T (Mercado Livre)	5,40
Roteador Multilaser N150 (Brasil)	60,00
Fonte Alimentação 15Vcc/ 1A	18,90
TOTAL INVESTIDO	**160,90**

Fonte: Autor.

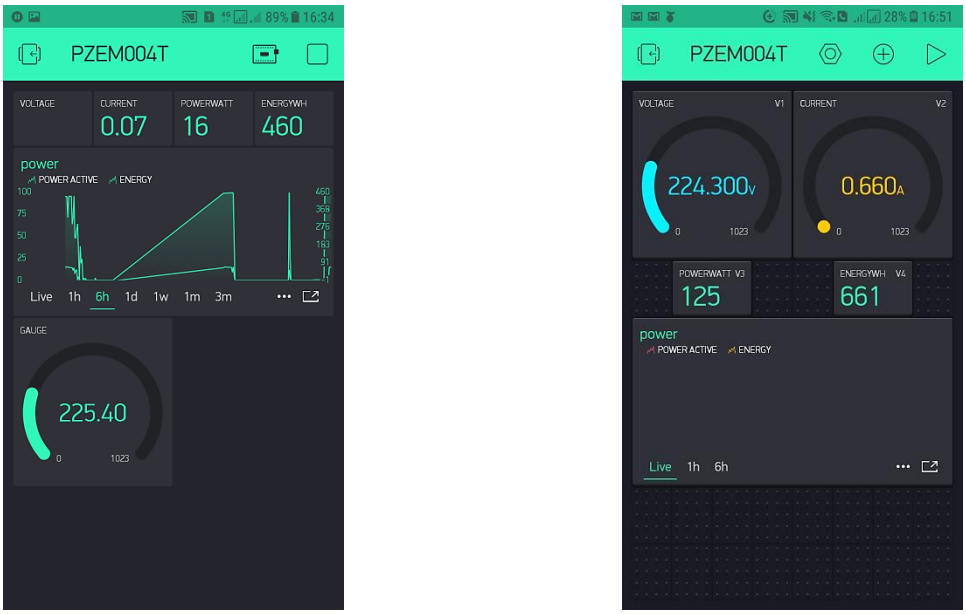
Considerando outros custos, pode se aproximar que o investimento necessário foi algo em torno de R$ 200,00 (duzentos reais). Para se ter uma ideia, os medidores da Peacefair chineses não baixam de R$ 40,00 (módulos completos, com caixinha de displays de 7 segmentos) e medidores similares, para barramento DIN tão pouco baixam de R$ 80,00 (lojas chinesas).

Levando em consideração que o roteador não faz parte do projeto, que fontes e placas podem ser simplificados e o próprio ESP pode ser montado em uma placa mais econômica, que nos valores estão incluídos frete e lucro dos vendedores, estima-se que é possível ter um protótipo em torno de R$ 100,00. Um produto comercial poderia sair ainda por menor valor de venda.

4.2. Captura de Tela dos Testes de Medição Usando Aplicativo Blynk

A sequência de capturas de tela da Figura 29 mostra a interface criada no Blynk para o projeto:

Figura 29 – Interface do projeto configurada Blynk rodando no smartphone. Em (a) primeiras configurações de testes, em (b) a aparência atual.



(a) (b)

Fonte: Autor.

Em (a) o exemplo de como a tela ficou inicialmente configurada durante os testes de comunicação com os pinos virtuais (V1, V2, V3 e V4). Em (b) como fica durante as leituras em tempo real quando foi testado em conjunto com o analisador de qualidade de energia (Fluke). Cada *widget* corresponde à leitura de um desses pinos (configurados no código do ESP).

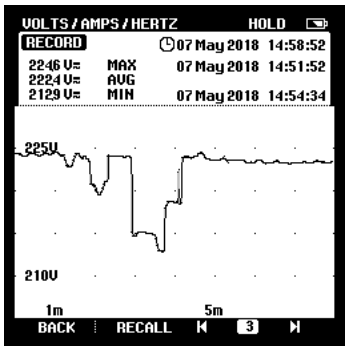
4.3. Verificando a Precisão das Leituras do PZEM004T com Analisador de Qualidade de Energia 43B Fluke.

É importante determinar o quão precisos são os valores de tensão e corrente retornados pelo PZEM especialmente em um ambiente residencial real na presença de cargas não lineares.

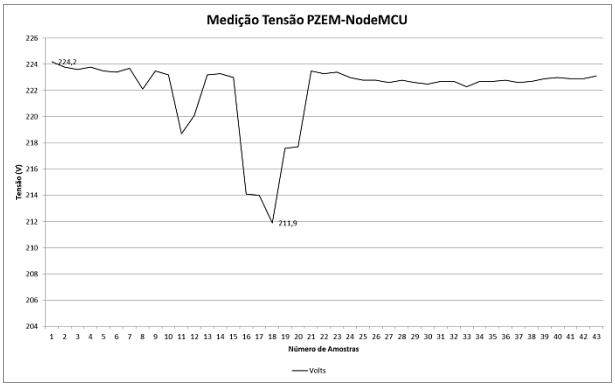
Foi utilizado o Analisador de Qualidade de Energia 43B da fabricante Fluke para comparar os valores do PZEM com os lidos por um instrumento calibrado e certificado.

Uma janela de amostrar de oito minutos aproximadamente foi coletada usando o analisador e o protótipo. As curvas obtidas podem ser vistas na Figura 30:

Figura 30 – Curvas comparativas de tensão extraídas do analisador da Fluke em comparação com dados coletados via protótipo, com janela aproximada de oito minutos.



(a)



(b)

Fonte: Autor.

O analisador foi configurado para:

- Medindo: Volt/Amperes/Hertz;
- Probe 1: Test Leads;
- Probe 2: 10mV/A.

Os relógios estavam configurados para data e hora locais, mas não estavam sincronizados.

Outro ponto importante trata da taxa de amostragem diferente entre o analisador e o protótipo, sendo deste último uma gravação no SD Card a cada cinco (5) segundos, o que faz com que nem a quantidade de pontos e nem a coincidência destes ocorra.

Mas esta é apenas uma comparação qualitativa e simples. Não é objetivo neste momento uma análise mais exaustiva em ambiente controlado, onde diversos parâmetros e variáveis poderiam ser ensaiados, o que será realizado na sequência deste trabalho.

4.4. Próximas Possibilidades de Continuidade para Finalização.

Como este trabalho ainda está em fase de avaliação antes de chegar à sua conclusão algumas soluções podem ainda ser abordadas:

- Testes e resultados utilizando cargas e fontes variáveis, dentro de um período padrão;
- Testes de outras plataformas e aplicativos, como o CloudMQTT e Cayenne, já citados;
- Finalização e possíveis melhorias no código do NodeMCU;
- Inserção ou melhorias no hardware do protótipo.

Esta seção será removida após as devidas revisões.

4.5. CRONOGRAMA DESTA FASE DO TRABALHO

A Tabela 8 mostra o cronograma do trabalho, tanto o atingido quanto o pretendido, até esta versão submetida.

Tabela 8: Cronograma de Atividades do Trabalho de Conclusão de Curso.

Etapa	**A concluir OU Concluído**																			
	MAR				**ABR**				**MAI**				**JUN**				**JUL**			
Definição e Entrega do Tema e Orientados	-	-	-	x																
Montagem e Ajuste do Protótipo Funcional	-	-	-	-	-	x														
Entrega do Esboço da Defesa Intermediária	-	-	-	-	-	-	-	-	-		?									
Defesa da Prévia do TCC	-	-	-	-	-	-	-	-	-			?								
Aplicação das Melhorias Propostas e Ajustes do Protótipo													?							
Escrita da Versão Final do TCC e Revisões															?					
Defesa Final do TCC																?				
Aplicação de Ajustes Finais																	?			

Fonte: Autor.

É esperado que após a conclusão da defesa prévia o restante dos pontos a serem concluídos, bem como as mudanças e alterações solicitadas pela banca sejam atendidas.

Esta seção será removida após as devidas revisões.

5. CONCLUSÕES

Neste trabalho foi exigido que houvesse disciplina, organização, foco, conhecimento de relações interpessoais, operar tradutores e boa dose de pesquisa até ter certeza que se estava entendendo ou indo pelo caminho correto.

É perceptível que para um produto comercial o protótipo cumpre apenas com fins didáticos e de aprendizado. Sua produção precisaria levar em conta refazer ou substituir diversos módulos por seus equivalentes CIs em versões mais baratas montados com placas específicas. Há visível sobra de recursos, que poderiam ser simplificados.

O módulo da Peacefair não é o tipo de placa classificado como “hardware aberto”, carecendo inclusive de disponibilidade de documentação, mas não se pode negar que é uma alternativa viável de baixo custo e que também inspira a criação de módulos compactos similares.

Embora ainda não mostrado neste trabalho, a análise visual dos resultados em tempo real durante os testes do protótipo (leituras no Serial Monitor, no Blynk, no SD Card e também no Fluke 43B) reportaram valores com uma precisão e velocidade de resposta até surpreendente, mostrando-se funcional e aplicável em teoria.

REFERÊNCIAS

ANEEL. **Energia no Brasil e no Mundo**. Agência Nacional de Energia Elétrica, Atlas de Energia Elétrica do Brasil, Box 2, Parte I], 2002. Disponível em: <http://www2.aneel.gov.br/arquivos/pdf/atlas_par1_cap2.pdf>. Acesso em 22 abr. 2018.

MME. **Resenha Energética Brasileira**. Ministério de Minas e Energia, 2015. Disponível em: <http://www.mme.gov.br/documents/1138787/1732840/Resenha+Energ%C3%A9tica+-+Brasil+2015.pdf/4e6b9a34-6b2e-48fa-9ef8-dc7008470bf2>. Acesso em 22 abr. 2018

EPE. **Consumo anual de energia elétrica por classe (nacional)** - 1995-2017. Empresa de Pesquisa Energética, 2017. Disponível em: < http://www.epe.gov.br/pt/publicacoes-dados-abertos/publicacoes/Consumo-Anual-de-Energia-Eletrica-por-classe-nacional>. Acesso em 22 abr. 2018.

ABESCO. **Desperdício de energia gera perdas de R$ 12,6 bilhões**. Associação Brasileira das Empresas de Serviços de Conservação de Energia, 2015. Disponível em: <http://www.abesco.com.br/pt/novidade/desperdicio-de-energia-gera-perdas-de-r-126-bilhoes>. Acesso em: 22 abr. 2018.

CUNHA, Joana. "**Desperdício consome 10% da energia elétrica no país, diz associação"**. Jornal Folha Mercado, 2015. Disponível em: <http://www1.folha.uol.com.br/mercado/2015/02/1586778-desperdicio-consome-10-da-energia-eletrica-no-pais-diz-associacao.shtml>. Acesso em 22 abr. 2018.

GUADANIN, Cláudia. **De 2011 a 2015, Brasil desperdiçou energia suficiente para um ano de consumo**. Jornal Gazeta do Povo, 2016. Disponível em: <http://www.gazetadopovo.com.br/economia/energia-e-sustentabilidade/de-2011-a-2015-brasil-desperdicou-energia-suficiente-para-um-ano-de-consumo-8bnk42j8ibd25of8e9yiwb5h1>. Acesso em: 22 abr. 2018.

NUWER, Rachel. "**Por que empresas de energia adoram medidores inteligentes?"**. Revista Exame [online], 2015. Disponível em: <https://exame.abril.com.br/tecnologia/por-que-empresas-de-energia-adoram-medidores-inteligentes>. Acesso em 22 abr. 2018.

HAYASHI, Ricardo. **Automação por meio de medidores inteligentes de energia elétrica permite adequação das distribuidoras à Tarifa Branca**. Editora Brasil Energia [online], 2018. Disponível em: <https://brasilenergia.editorabrasilenergia.com.br/artigo-automacao-por-meio-de-medidores-inteligentes-de-energia-eletrica-permite-adequacao-das-distribuidoras-a-tarifa-branca>. Acesso em 22 abr. 2018.

WEG. **Medidores inteligentes de energia WEG no maior projeto de Smart Grid do Brasil**. WEG Energia, 2016. Disponível em:

<https://www.weg.net/institutional/BR/pt/news/produtos-e-solucoes/medidores-inteligentes-de-energia-weg-no-maior-projeto-de-smart-grid-do-brasil>. Acesso em 22 abr. 2018.

SDIC. **Energy Measurement SOC SD3004**. SDICMICRO, 2012. Disponível em: <http://www.sdicmicro.com/DataSheet/SD3004%20datasheet%20v0.2c.pdf>. Acesso em 22 abr. 2018.

EVANS, Dave. **A Internet das Coisas: Como a próxima evolução da Internet está mudando tudo**. White Papper da Cisco, 2011. Disponível em: <https://www.cisco.com/c/dam/global/pt_br/assets/executives/pdf/internet_of_things_iot_ibsg_0411final.pdf>. Acesso em 23 abr. 2018.

JAVED, Adeel. **Criando Projetos com Arduino para a Internet das Coisas**. São Paulo: Novatec, 2017.

ALECRIN, Emerson. **Software livre, código aberto e software gratuito: as diferenças**. InfoWester, 2013. Disponível em: <https://www.infowester.com/freexopen.php>. Acesso em 23 abr. 2018.

THOMSEN, Adilson. "**Vamos falar de Open Hardware?**". FilipiFlop, 2014. Disponível em: <https://www.filipeflop.com/blog/open-hardware-livre/>. Acesso em 23 abr. 2018.

MANCINI, Mônica. **Internet das Coisas: História, Conceitos, Aplicações e Desafios.** USP, 2017. Disponível em: <https://pmisp.org.br/documents/acervo-arquivos/241-internet-das-coisas-historia-conceitos-aplicacoes-e-desafios/file>. Acesso em 24 abr. 2018.

ESPRESSIF. **ESP8266EX Datasheet**. Version 5.8, 2018 [English]. Disponível em: <https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>. Acesso em 24 abr. 2018.

SOKOLOV, Oleg. **Arduino communication library for Peacefair PZEM-004T Energy Monitor**. GitHub, 2018. Disponível em: <https://github.com/olehs/PZEM004T>. Acesso em 27 abr. 2018.

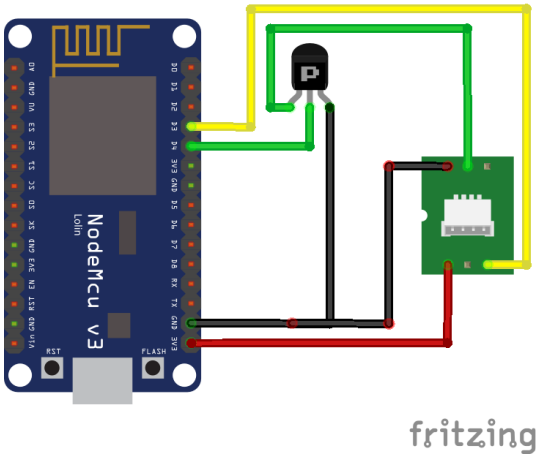
OLIVEIRA, Sérgio de. **Internet das Coisas com ESP8266, Arduino e Raspbery Pi**. São Paulo: Novatec, 2017.

APÊNDICES

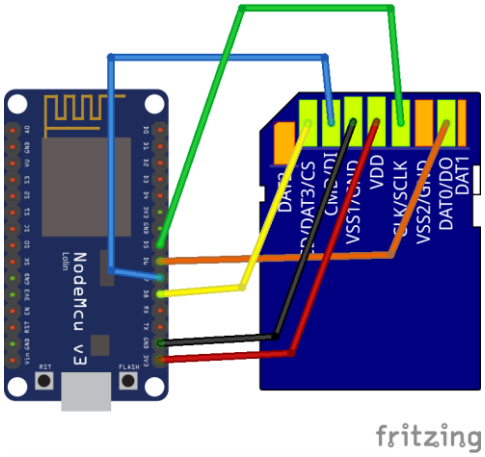
Elemento opcional. Colocado após o glossário e constituído de informações elaboradas pelo autor do trabalho, não incluídas no texto. Os apêndices são identificados por letras maiúsculas consecutivas, travessão e pelos respectivos títulos.

ANEXOS

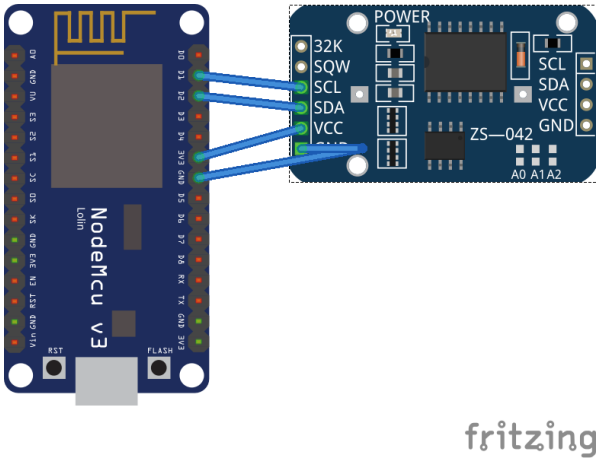
A. — Diagramas de Fiação



Fiação PZEM004T

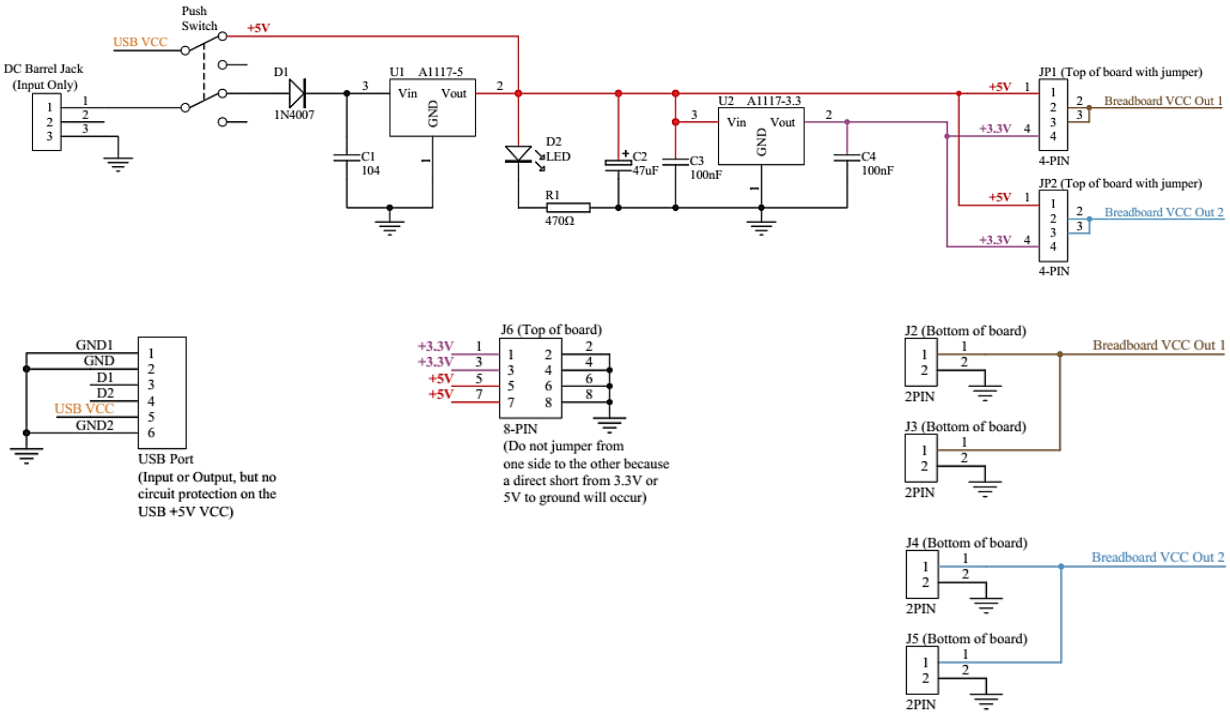


Fiação SD Card



Fiação RTC DS3231

B. — Diagrama Esquemático do Módulo de Alimentação YWROBOT



C. — Código Fonte Completo Comentado

```
/*CABEÇALHO DE IDENTIFICAÇÃO AUTORAL
 * Autor: Gerson E O Sena
 * Projeto: Medidor Datalogger de Energia Elétrica OpenSource com NodeMCU Lolin
 * Data: 2018
 * Local: Unipampa, Alegrete, RS, Brasil
 * Contribuições: Comunidade OpenSource, desenvolvedores, autores de artigos WEB.
 */

/*============================================================================================
==========================
 * INCLUSÃO DAS BIBLIOTECAS NECESSÁRIAS
 *
 */
#include <PZEM004T.h>       //Biblioteca exclusiva do PZEM004T escrita para Arduino
#include <SD.h>             //Biblioteca de comunicação com o cartão SD nativa
#include <Wire.h>           //Biblioteca de comunicação I2C nativa
#include "RTClib.h"         //Biblioteca de comunidação RTC que funcionou com Arduino/
ESP
//#include <ESP8266WiFi.h>    //Biblioteca nativa do ESP para conexões WiFi


/*Comunicação I2C: RTC, PCF8574.
 * D1 (05) = SCL
 * D2 (04) = SDA
 * Endereços scaneados:
 * 0x3F = PCF8574
 * 0x57 = RTC DS3231
 * 0x68 = ? (pode ser algo na própria placa do NodeMCU
 *
 * Comunicação SPI: SD Card
 * D8 (15) = CS
 * D7 (13) = MOSI
 * D6 (12) = SCK
 * D5 (14) = MISO
 */


/*============================================================================================
==========================
 * DEFINIÇÃO DAS VARIÁVEIS GLOBAIS
 */
//PZEM
PZEM004T pzem(0, 2);          // D3 (GPIO00) = RX (recebe TX do PZEM), D4 (GPIO02) = TX
(recebe RX do Pzem pelo Q 2N2907)
IPAddress ip(192,168,1,1);    // IP de comunicação do PZEM. Inicializa a comunicação com
o módulo
bool pzemrdy = false;         // Variável tipo bolean para identificar se a comunicação
com o ESP teve sucesso

//SD
const int chipSelect = 4;     // Valor atribuído ao Pino CS do SD card
bool CardOk = true;           // Variável tipo bolean para identificar se a comunicação
com o SD teve sucesso
File dataFile;                // Cria objeto responsável por ler/ escrever no SD

//RTC
RTC_Millis rtc;               // Cria o objeto rtc associado ao chip DS3231

float v, i, p, e, maior, menor, x, y, z = 0; // Variáveis para armazenar e tratar
leituras do PZEM
char daysOfTheWeek[7][12] = {"DOM", "SEG", "TER", "QUA", "QUI", "SEX", "SÁB"};  // Cria
array nomes e posição dos dias da semana


/*============================================================================================
==========================
 * SETUP, INICIALIZAÇÃO DAS VARIÁVEIS, FUNÇÕES E BIBLIOTECAS
 */

void setup() {
  Serial.begin(115200);
  Wire.begin();

//RTC
```

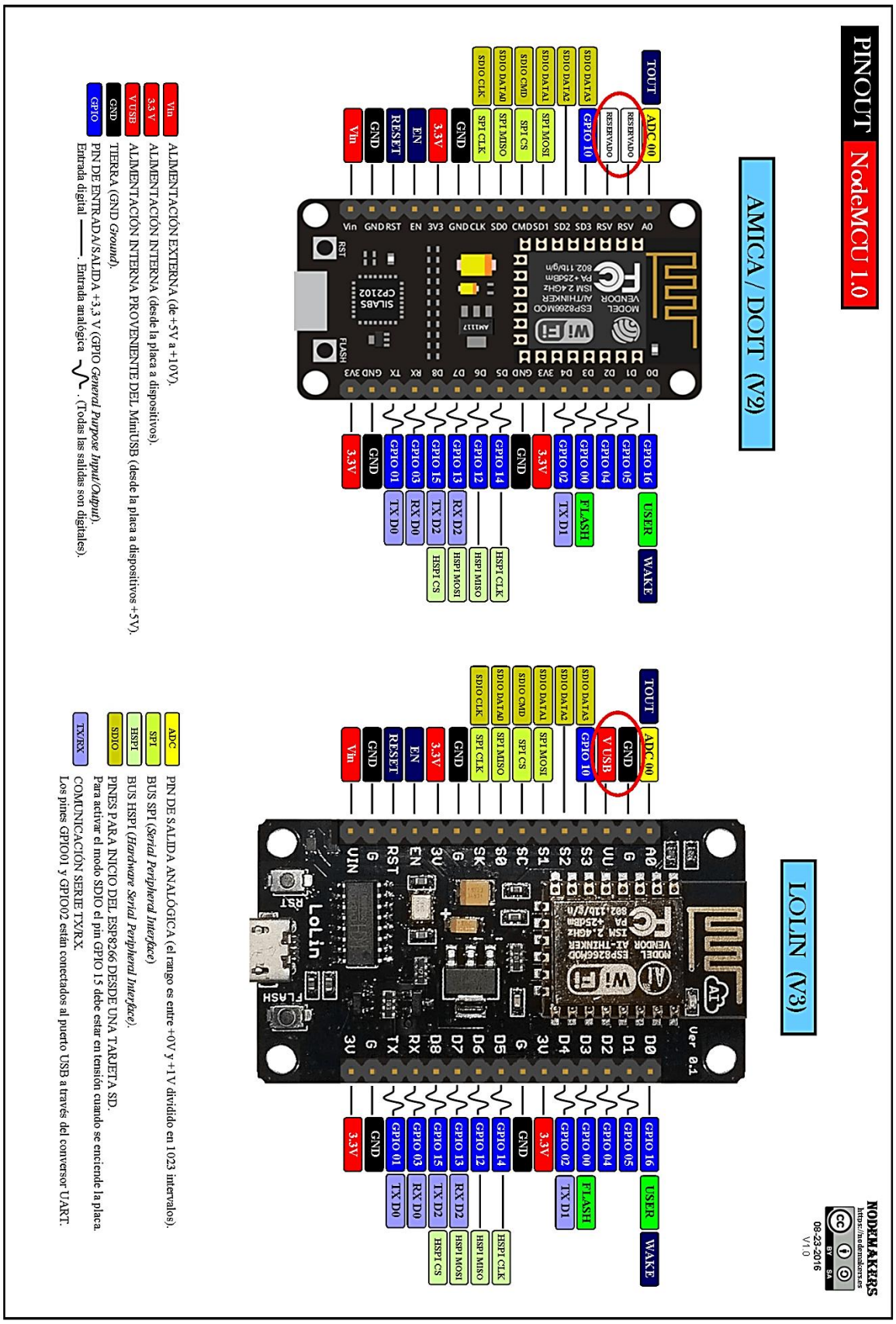
```
75  rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));   // Configura a Data e Hora do RTC pela
76  do SISTEMA no momento da compilação
77  //rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));       // Configura a Data e Hora do RTC
78  MANUALMENTE
79
80  //PZEM
81    while (!pzemrdy) {                        // Testa e força a comunicação com o PZEM
82        Serial.println("Connecting to PZEM...");
83        pzemrdy = pzem.setAddress(ip);
84        delay(1000);
85     }
86
87  //SD
88    if (!SD.begin(chipSelect)) {              // Testa comunicação com o SD
89      Serial.println("Erro na leitura do arquivo, não existe um SD Card ou Módulo
90  incorretamente conectado");
91      CardOk = false;
92      return;
93    }
94
95    if (CardOk) {                             // Caso SD esteja esteja OK, cria o arquivo
96  'datalog.csv' em modo ESCRITA
97      dataFile = SD.open("datalog.csv", FILE_WRITE);
98      Serial.println("SD Card OK!");
99    }
100
101 String leitura = "";                        // Limpa o campo string que será armazenado
102 no SD como .csv
103 delay(1000);
104
105 leitura = String("DATA") + ";" + String("DSEM") + ";" + String("HORA") + ";"
106           + String("Volts") + ";" + String("Amps") + ";" + String("POTi") + ";" +
107 String("Energy");
108 if (dataFile) {                          // Continuar a escrever no arquivo, mas 'zeros'
109       Serial.println("Escrevendo CABEÇALHO...");
110       dataFile.println(leitura);            // Escrevemos no arquivos e pulamos uma linha
111       dataFile.close();                     // Fechamos o arquivo
112       Serial.println("Done");
113       }
114     return;
115
116 }
117
118
119 /*=============================================================================================
120 ============================
121  * LAÇO DE EXECUÇÃO DAS ROTINAS
122  */
123 void loop() {
124 //SD
125   if (CardOk) {                             // Caso SD esteja esteja OK, cria o arquivo
126 'datalog.csv' em modo ESCRITA
127     dataFile = SD.open("datalog.csv", FILE_WRITE);
128     Serial.println("Cartão SD inicializado para escrita...");
129   }
130
131 String leitura = "";                        // Limpa o campo string que será armazenado
132 no SD como .csv
133 delay(1000);
134
135 //RTC
136 DateTime now = rtc.now();                   // Registra o tempo do RTC quando a rotina
137 passa por essa linha
138
139 delay(3000);
140     Serial.print(now.year(), DEC); Serial.print('/'); Serial.print(now.month(), DEC);
141 Serial.print('/'); Serial.print(now.day(), DEC);
142     Serial.print(" ("); Serial.print(daysOfTheWeek[now.dayOfTheWeek()]); Serial.print(")
143 ");
144     Serial.print(now.hour(), DEC); Serial.print(':'); Serial.print(now.minute(), DEC);
145 Serial.print(':'); Serial.print(now.second(), DEC);
146     Serial.print("; ");
147
148 //PZEM
149 /* Tratamento de valores espúrios (a lib do PZEM foi escrita para arquitetura AVR, não
150 ESP
```

```
 *  Armazena 3 valores em sequência e prepara para comparação/ limpeza de valores espúrios
 */
//Leitura Tensão para Testar FALTA
x = pzem.voltage(ip); y = pzem.voltage(ip); z = pzem.voltage(ip);
maior = x; menor = x;
  /*
   * LAÇO CONDICIONAL PRINCIPAL
   */
  if (x < 0.0 && y < 0.0 && z < 0.0) {      // Laço que identifica e confirma falta de energia x = y = z = -1 do PZEM
    v = 0.0; i = 0.0; p = 0.0; e = 0.0;     // Zera todos os valores para não 'printar' código '-1'
    Serial.print(v); Serial.println("V FALTA");

    //SD *** String linhas com zeros, para facilitar plotar curvas de carga
    leitura = String(now.year(), DEC) + "/" + String(now.month(), DEC) + "/" + String(now.day(), DEC) + ";"
    + String(daysOfTheWeek[now.dayOfTheWeek()]) + ";"
    + String(now.hour(), DEC) + ":" + String(now.minute(), DEC) + ":" + String(now.second(), DEC) + ";"
    + String(v) + ";" + String(i) + ";" + String(p) + ";" + String(e) + ";";

    if (dataFile) {                         // Continuar a escrever no arquivo, mas 'zeros'
      Serial.println("Escrevendo FALTA...");
      dataFile.println(leitura);            // Escrevemos no arquivos e pulamos uma linha
      dataFile.close();                     // Fechamos o arquivo
      Serial.println("Done");
      }
    return;
  }
  else {                                    // Leitura de valores PZEM em Regime (sem FALTA)
    espurgov(pzem.voltage(ip), pzem.voltage(ip), pzem.voltage(ip));
    espurgoi(pzem.current(ip), pzem.current(ip), pzem.current(ip));
    espurgop(pzem.power(ip), pzem.power(ip), pzem.power(ip));
    espurgoe(pzem.energy(ip), pzem.energy(ip), pzem.energy(ip));
      Serial.print(v);Serial.print("W; ");
      Serial.print(i);Serial.print("W; ");
      Serial.print(p);Serial.print("W; ");
      Serial.print(e);Serial.println("Wh; ");
  }

// Caso os valores lidos do sensor não sejam válidos
  if (isnan(v) || isnan(i) || isnan(p) || isnan(e)) {
    Serial.println ("Falha na leitura do sensor");
    delay(1500);
    return;
  }

//SD
// SE tudo estiver certo com os dados, cria a string com os valores lidos
// SD *** String linhas com zeros, para facilitar plotar curvas de carga
    leitura = String(now.year(), DEC) + "/" + String(now.month(), DEC) + "/" + String(now.day(), DEC) + ";"
    + String(daysOfTheWeek[now.dayOfTheWeek()]) + ";"
    + String(now.hour(), DEC) + ":" + String(now.minute(), DEC) + ":" + String(now.second(), DEC) + ";"
    + String(v) + ";" + String(i) + ";" + String(p) + ";" + String(e) + ";";

  if (dataFile) {
    Serial.println("Escrevendo...");
    dataFile.println(leitura);  // Escrevemos no arquivos e pulamos uma linha
    dataFile.close();           // Fechamos o arquivo
    Serial.println("Done");
  }

  delay(1500);

}


/*=============================================================================================
==========================
 * BLOCO DE FUNÇÕES UTILIZADAS
```

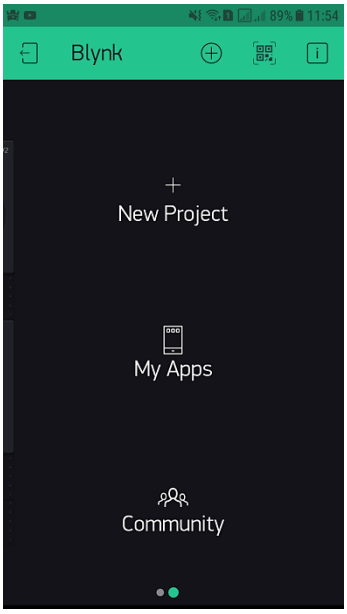
```
227   * Funções de espurdo dados quebrados
228   */
229  // Leitura e Tratamento dos valores de Tensão
230  void *espurgov(float x, float y, float z){
231    float maior = x; float menor = x;
232        if (y > menor){
233          menor = y;
234          }
235          else {
236            menor = 0.0;
237          }
238          if (y > maior && y >= 0.0){
239            maior = y;
240            }
241            if (z > menor){
242              menor = z;
243              }
244            else {
245              menor = 0.0;
246              }
247              if (z > maior && z >= 0.0){
248                maior = z;
249                }
250        v = maior;
251  }
252
253  // Leitura e Tratamento dos valores de Corrente
254  void *espurgoi(float x, float y, float z){
255    float maior = x; float menor = x;
256        if (y > menor){
257          menor = y;
258          }
259          else {
260            menor = 0.0;
261          }
262          if (y > maior && y >= 0.0){
263            maior = y;
264            }
265            if (z > menor){
266              menor = z;
267              }
268            else {
269              menor = 0.0;
270              }
271              if (z > maior && z >= 0.0){
272                maior = z;
273                }
274        i = maior;
275  }
276
277  // Leitura e Tratamento dos valores de Potência Instantânea
278  void *espurgop(float x, float y, float z){
279    float maior = x; float menor = x;
280        if (y > menor){
281          menor = y;
282          }
283          else {
284            menor = 0.0;
285          }
286          if (y > maior && y >= 0.0){
287            maior = y;
288            }
289            if (z > menor){
290              menor = z;
291              }
292            else {
293              menor = 0.0;
294              }
295              if (z > maior && z >= 0.0){
296                maior = z;
297                }
298        p = maior;
299  }
300
301  // Leitura e Tratamento dos valores de Energia Consumida
302  void *espurgoe(float x, float y, float z){
```

```
303   float maior = x; float menor = x;
304       if (y > menor){
305         menor = y;
306         }
307         else {
308           menor = 0.0;
309         }
310         if (y > maior && y >= 0.0){
311           maior = y;
312           }
313           if (z > menor){
314             menor = z;
315             }
316           else {
317             menor = 0.0;
318             }
319             if (z > maior && z >= 0.0){
320               maior = z;
321               }
322       e = maior;
323 }
```

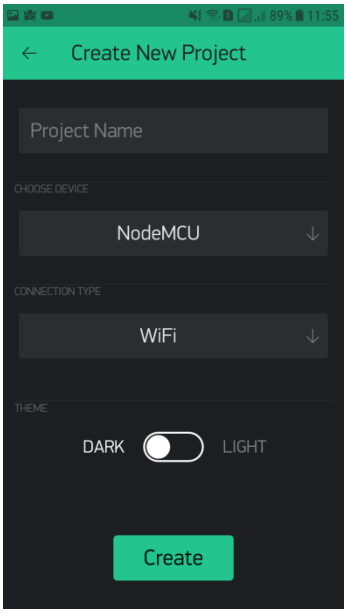
D. — Pinout do NodeMCU Lolin



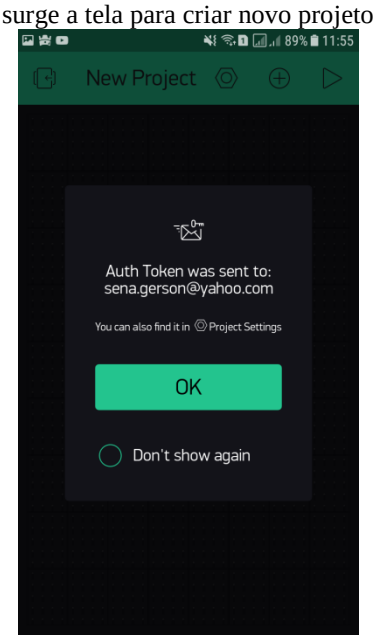
E. — Configuração Inicial Básica do Aplicativo Blynk no Smartphone.



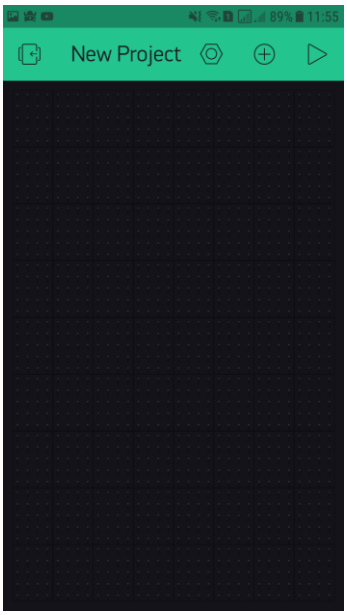
Após fazer acessar o app com usuário e senha criados, surge a tela para criar novo projeto



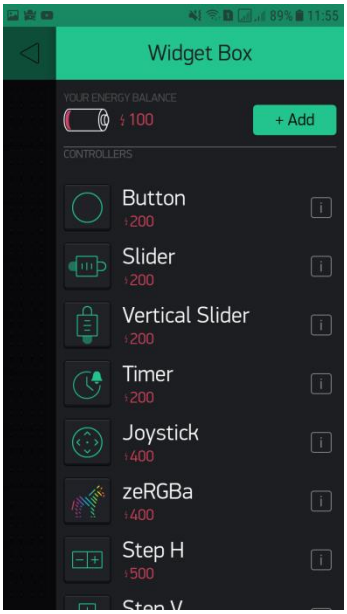
Ao clicar em New Project, a tela que surge é esta.



Um código “token” será gerado e enviado ao e-mail cadastrado. Clique em “OK”. Abra seu e-mail (pode ter ido parar em ‘spam’) e copie o a chave token (vai usar no código inserido no ESP)



Uma área de trabalho em branco será exibida, e no botão + widgets serão mostrados. Após adicionar algum widget este pode ser arrastado pela tela.



Alguns exemplos de widgets disponíveis. Observe que por já ter criado um projeto, o ícone da bateria logo acima não consta 2000, mas sim apenas 100, pois já usei 1900. Se precisar de mais basta clicar em +Add e comprar.

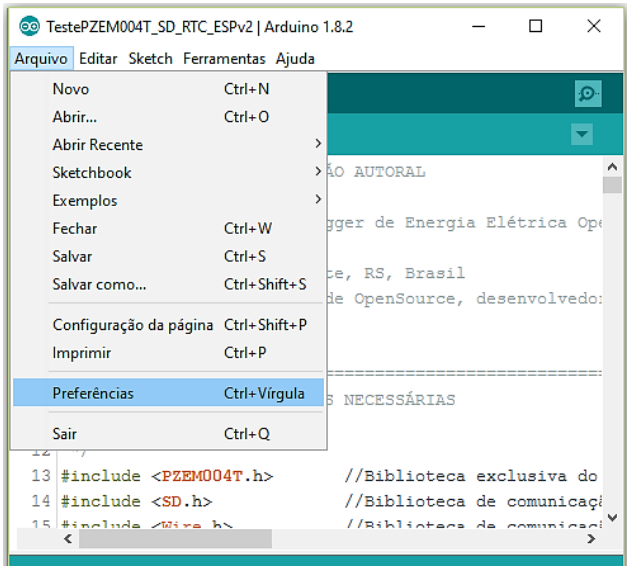
Aqui um exemplo de como a tela ficou inicialmente configurada durante os testes de comunicação com os pinos virtuais. Cada widget corresponde a leitura de um desses pinos (setados no código do ESP).

F. — Usando o ESP8266 com a IDE Arduino

Nesta seção será mostrado o que é necessário para possibilitar isso, configurando a IDE.

A primeira tarefa é abrir a IDE. Isto feito o primeiro passo é o da Figura 31:

Figura 31 – Passo 1 – Configuração IDE Arduino.

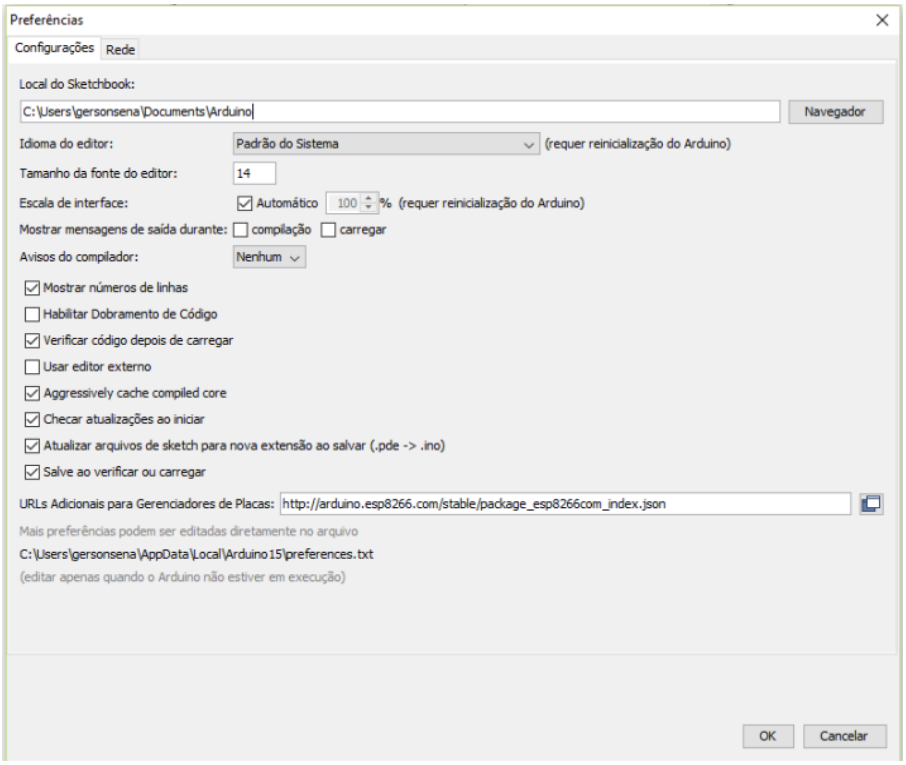


Fonte: Autor.

Na janela da Figura 32 insere-se o endereço[30] do repositório a ser buscado para adição das bibliotecas dos módulos que permitirão à IDE reconhecer e gravar nos modelos de ESP:

[30] Link do repositório: http://arduino.esp8266.com/stable/package_esp8266com_index.json

Figura 32 – Passo 2 – Configuração IDE Arduino, repositório.

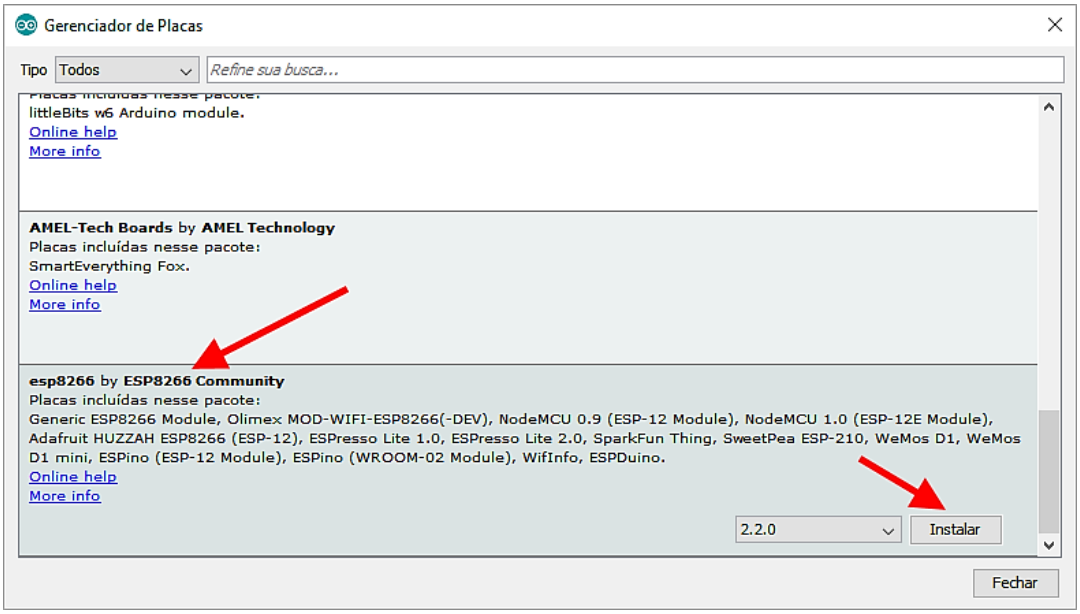


Fonte: Autor.

Basta clicar em “OK”.

É preciso então fazer a instalação da biblioteca que agora surgirá no gerenciador de bibliotecas, como mostrado na Figura 33:

Figura 33 – Passo 3 – Configuração IDE Arduino, instalando a biblioteca ESP.



Fonte: Autor.

Agora, basta conectar a placa do ESP numa das portas USB do computador e verificar se ela foi reconhecida procurando no Menu: "*Ferramentas > Placa: ...*", e depois em: "*Ferramentas > Porta: ...*".

Selecionadas tanto a placa como a porta USB corretas, basta agora carregar o sketch desejado na IDE e fazer a compilação para o ESP. Lembrando que todas as bibliotecas listadas pelo código precisam ter sido instaladas previamente.

Não há espaço neste trabalho para ensinar todos os detalhes dessas configurações, mas a Internet tem uma infinidade de publicações ensinando a fazer e resolver essas etapas.

G. — Exemplo de conversão dos comandos AT usados no PZEM004T

...