

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ENGENHARIA ELETRÔNICA

LUCAS SIMONI PETRELLI

**REDE DE PRODUTOS CHAVEADORES INTELIGENTES:
PROJETO E IMPLEMENTAÇÃO**

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2018

LUCAS SIMONI PETRELLI

**REDE DE PRODUTOS CHAVEADORES INTELIGENTES:
PROJETO E IMPLEMENTAÇÃO**

Trabalho de Conclusão de Curso de graduação do curso de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para a obtenção do grau de Engenheiro.

Orientador: Prof. Dr. Hugo Vieira Neto

CURITIBA
2018

LUCAS SIMONI PETRELLI

REDE DE PRODUTOS CHAVEADORES INTELIGENTES
Projeto e implementação

Este Trabalho de Conclusão de Curso de Graduação foi apresentado como requisito parcial para obtenção do título de Engenheiro Eletrônico, do curso de Engenharia Eletrônica do Departamento Acadêmico de Eletrônica (DAELN) outorgado pela Universidade Tecnológica Federal do Paraná (UTFPR). Lucas Simoni Petrelli foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Curitiba, 24 de maio de 2018.

Prof. Dr. Robinson Vida Noronha
Coordenador de Curso
Engenharia Eletrônica

Prof^ª. Dr^ª. Carmen Caroline Raserá
Responsável pelos Trabalhos de Conclusão de Curso
de Engenharia Eletrônica do DAELN

BANCA EXAMINADORA

Prof. Dr. Hugo Vieira Neto
Universidade Tecnológica Federal do Paraná
Orientador

Prof. Dr. Luiz Copetti
Universidade Tecnológica Federal do Paraná

Prof. Dr. André Eugênio Lazzaretti
Universidade Tecnológica Federal do Paraná

A folha de aprovação assinada encontra-se na Coordenação do Curso de Engenharia Eletrônica.

RESUMO

PETRELLI, Lucas Simoni. Rede de produtos chaveadores inteligentes: projeto e implementação. 2018. 69 f. Trabalho de Conclusão de Curso (Graduação) – Engenharia Eletrônica. Universidade Tecnológica Federal do Paraná. Curitiba, 2018.

Buscando aproveitar do cenário tecnológico atual, no qual cada vez mais dispositivos tendem a ser conectados em uma rede inteligente, o autor propõe o projeto e a implementação de um produto que busca automatizar a energização dos aparelhos eletroeletrônicos nas casas, escritórios e propriedades de seus usuários. São levantados requisitos técnicos e funcionais do produto final, então são realizadas as implementações de hardware, software embarcado e software para páginas web a fim de se gerar um protótipo que atinja as funcionalidades propostas. Nesse documento são discutidas as diversas escolhas feitas e problemas encontrados, finalizando com propostas de melhorias para que o protótipo desenvolvido se torne um produto comercial.

Palavras-chave: Internet das coisas. Sistemas embarcados. Automação residencial. Engenharia de produto.

ABSTRACT

PETRELLI, Lucas Simoni. Network of smart switching devices: design and implementation. 2018. 69 p. Undergraduate thesis – Electronics Engineering. Federal University of Technology – Paraná. Curitiba, 2018.

Seeking to take part in the current technology scenario, in which more and more devices tend to be connected through a smart network, the author proposes the design and implementation of a product that aims to automate the energization of the devices available in the homes, offices and other properties of its users. The technical and functional requirements of the product are set, followed by the description and implementation of the related hardware, embedded software and web pages in order to generate a prototype that has the proposed features. In this document, various decisions that were made and problems that were encountered are discussed, culminating in a discussion of improvements needed to turn the mentioned prototype into a commercial product.

Keywords: Internet of things. Embedded systems. Home automation. Product engineering.

SUMÁRIO

| | | |
|------|---|----|
| 1. | Introdução..... | 7 |
| 2. | Objetivos | 9 |
| 3. | Projeto..... | 9 |
| 3.1. | Descrição dos produtos | 11 |
| 3.2. | Requisitos..... | 11 |
| 3.3. | Casos de uso..... | 13 |
| 4. | Implementação | 15 |
| 4.1. | Escolhas realizadas..... | 15 |
| 4.2. | Tecnologias utilizadas | 18 |
| 4.3. | Projeto de hardware | 23 |
| 4.4. | Projeto de comportamento e de rede | 31 |
| 4.5. | Projeto de software | 38 |
| 4.6. | Projeto web..... | 56 |
| 5. | Discussão dos resultados..... | 61 |
| 5.1. | Pontos fortes..... | 62 |
| 5.2. | Comparação com produtos já existentes | 62 |
| 5.3. | Problemas encontrados | 63 |
| 5.4. | Melhorias vislumbradas..... | 64 |
| | REFERÊNCIAS..... | 66 |
| | APÊNDICE A – Código fonte e referências externas..... | 68 |

1. INTRODUÇÃO

Este documento versa sobre o desenvolvimento dos protótipos de uma linha de produtos de automação residencial conectados entre si via uma rede sem fio, aproveitando das tecnologias disponíveis no mercado e buscando atingir as expectativas do consumidor por um produto de interface amigável.

Com a crescente automação de utensílios e de processos que o mundo vivencia através do uso de microcontroladores capazes de gerar e processar informações de modo conectado – base do conceito de IoT (*Internet of Things*, internet das coisas) – buscou-se criar um produto de funcionamento simples que possibilite o controle da energização de utensílios elétricos, a fim de providenciar ao cliente comum – dono de residência, prédio comercial ou escritório – um contato com a automação.

Não existindo ainda soluções de produtos conectados populares no Brasil, e considerando o alto custo de importação de soluções parecidas do exterior (linha WeMo da fabricante Belkin, linha Hue da Philips, produtos da Insteon ou da D-Link) – soluções estas que em sua maioria necessitam de um dispositivo roteador ou concentrador dedicado de propriedade da marca para exercer todas as suas funcionalidades – vislumbrou-se a necessidade de um dispositivo de baixo custo, manuseio simples, capacidade de funcionamento em rede sem concentrador e de interface multiplataforma (PC, smartphone e tablet).

Na comunidade acadêmica os seguintes trabalhos tem destaque devido às semelhanças com o projeto descrito por este documento: Automação Residencial de Baixo Custo por Meio de Dispositivos Móveis com Sistema Operacional Android (BEGHINI, 2013), que descreve a implementação de uma rede de automação residencial baseada na plataforma Arduino utilizando da tecnologia Ethernet; Implementação de um Sistema de Automação Residencial Modular Sem Fio (ALMEIDA, 2009), onde é descrito o desenvolvimento e a construção de um sistema de automação em rede RF; Ambos os projetos referenciados versam sobre a implementação de uma rede de automação controlada pelo usuário, porém diferem da proposta do projeto aqui descrito devido a necessidade de uma central e do meio de comunicação utilizado.

Utilizando dos conhecimentos de hardware, software, sistemas embarcados e de projeto de produto obtidos ao longo do curso de Engenharia Eletrônica em conjunto com as práticas vivenciadas durante o período de estágio foram realizados o projeto e a implementação de um protótipo funcional. São

discutidas as características do dispositivo a ser implementado, os requisitos funcionais que ele deve cumprir, as tecnologias utilizadas e as construções de hardware e software realizadas. Ao final são discutidos os pontos fortes da implementação realizada, semelhanças e diferenças com alguns produtos já disponíveis no mercado e as possíveis melhorias a fim de se obter um produto comercial a partir do protótipo finalizado.

2. OBJETIVOS

O objetivo geral deste projeto é entregar um conjunto de protótipos eletrônicos, alinhados ao conceito de IoT, que controlem a alimentação de utensílios eletroeletrônicos de maneira automática seguindo uma programação do usuário.

Como objetivos específicos buscou-se aplicar os conhecimentos adquiridos ao longo do curso de engenharia e de estágios realizados a fim de se atingir o objetivo geral. São os objetivos específicos:

- Aplicar os procedimentos para a realização de um projeto – levantamento de requisitos, descrição da implementação – fundamentadas em Engenharia de Produto e Sistemas Embarcados;
- Construir um protótipo de hardware, aplicando-se os conhecimentos adquiridos em disciplinas como Eletrônica Básica, Semicondutores de Potência e Fabricação Eletrônica a fim de se obter a placa de circuito impresso dos protótipos.
- Criar um software coeso e desacoplado, aplicando-se os conhecimentos adquiridos nas disciplinas Fundamentos de programação 1 e 2, Estruturas de Dados;
- Fazer uso de um sistema operacional, baseando-se nos benefícios vislumbrados nas disciplinas Sistemas Operacionais e Sistemas Embarcados;
- Realizar o projeto do comportamento em rede dos protótipos, utilizando tecnologias e topologias vistas em Redes de Computadores;
- Entregar um protótipo estável através de testes ao longo do desenvolvimento, seguindo boas práticas aprendidas em aulas de laboratório e durante o estágio supervisionado.
 - Entende-se por protótipo estável o conjunto composto por: uma placa de circuito impresso sem sobreaquecimento de componentes e com conectores apropriados para cada tipo de conexão; e um software que consiga ser executado sem estouros de memória, corrupção de dados ou travamentos devido à disputa por recursos (*deadlocks*).

3. PROJETO

O projeto contempla três artefatos distintos: uma tomada, um interruptor – ambos com conectividade sem fio para seu controle – e uma interface web. Os produtos devem ser compactos o suficiente para caber nas cavidades de tomadas e interruptores já disponíveis nas construções, de modo a substituir os existentes.

Após instalados, os produtos possibilitam ao usuário o controle através da interface web sobre o fornecimento de energia aos aparelhos eletroeletrônicos de sua propriedade. Com o usuário conectado à mesma rede é possível configurar horários em que os aparelhos devam ser ligados, bem como programar dinâmicas automáticas em resposta aos eventos detectados que ocorrerem no ambiente onde os produtos estão instalados.

O produto apela ao consumidor pela sua simplicidade de instalação e por sua escalabilidade sem a necessidade de dispositivos auxiliares. Tudo o que o consumidor necessita é um ponto de energia e um roteador wi-fi para que o produto entregue suas funcionalidades, sem a necessidade de roteadores ou concentradores.

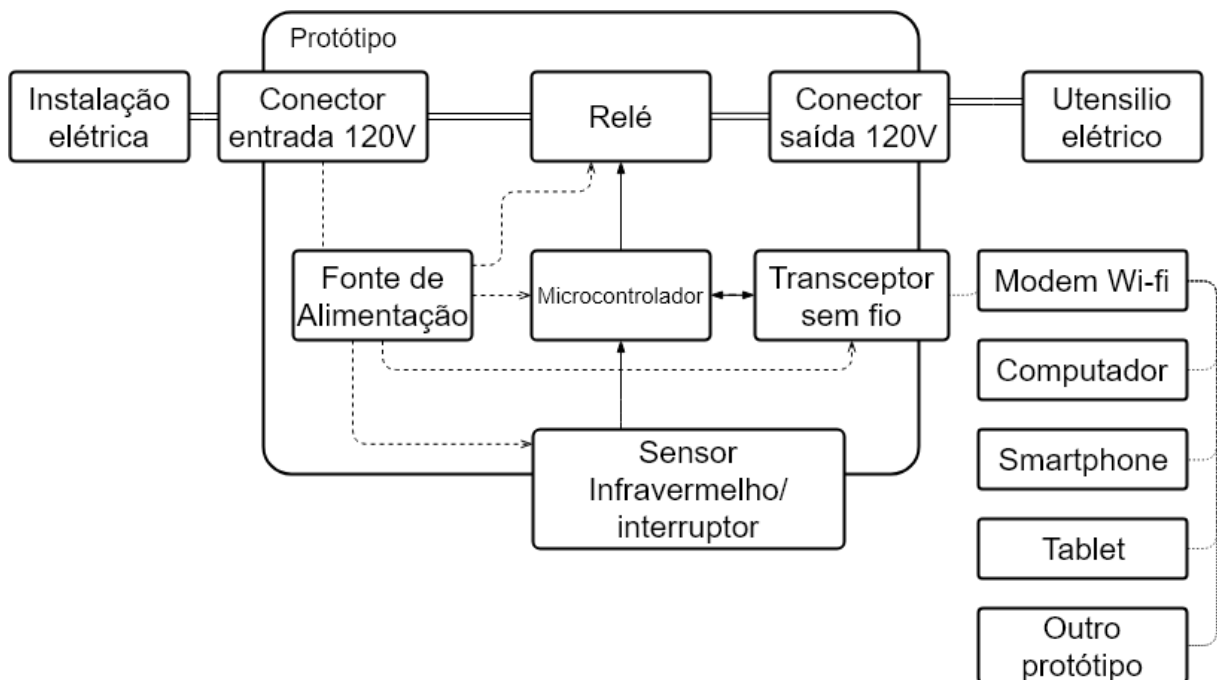


Figura 1: Diagrama em blocos resumido da solução
Fonte: autoria própria

A Figura 1 demonstra o diagrama em blocos da solução proposta, evidenciando o protótipo desenvolvido com elementos externos que a ele se conectam. Através de um conector o protótipo recebe energia da instalação elétrica da rede. Esta energia é chaveada por um relé, que controla a entrega de energia a

um utensílio elétrico conectado a um conector de saída no protótipo. A energia do conector de entrada também é utilizada internamente pelos componentes do protótipo, sendo necessária uma fonte de alimentação para adequar os níveis de tensão de acordo com aqueles requisitados por cada componente. O microcontrolador é encarregado do controle dos demais blocos do protótipo, acionando o relé, verificando o estado do sensor e utilizando o transceptor sem fio para se comunicar com outros dispositivos através de um modem sem fio.

3.1. Descrição dos produtos

3.1.1. Dispositivos materiais

A tomada eletrônica e interruptor eletrônico são entendidos como dispositivos materiais do projeto. A composição dos dispositivos é a seguinte:

- Uma placa de circuito impresso abrigando os componentes necessários para se alcançar os requisitos da seção 3.2 (requisitos funcionais);
- Uma conexão de alimentação elétrica, com bornes para a conexão dos fios fase e neutro da instalação elétrica onde o produto será instalado;
- Uma conexão de alimentação controlada, utilizada para se conectar um aparelho eletroeletrônico cuja alimentação deseja-se controlar;
- Um elemento de interação com o ambiente – sensor infravermelho no caso da tomada e chave estacionária no caso do interruptor.

3.1.2. Interface

Uma interface que disponibilize ao usuário meios de verificar e alterar a configuração de todos os produtos instalados conectados à rede sem fio local. Deve ser acessível através de um computador, *tablet* ou *smartphone* conectado à mesma rede que os dispositivos materiais.

3.2. Requisitos

Os requisitos aqui descritos servem como guias para a implementação do produto, delimitando as atividades que o sistema final deve ou não desempenhar. Toma-se que aquilo que não é proposto na descrição dos requisitos fica em aberto, de modo a favorecer sempre a simplificação da implementação.

A implementação é descrita no Capítulo 4, enquanto a discussão do relacionamento entre a realização dos requisitos e a implementação é discutida no Capítulo 5. Os casos de uso da Seção 3.3 descrevem situações onde os requisitos funcionais são utilizados.

3.2.1. Requisitos Funcionais

RF1. Os produtos devem ser capazes de ligar ou desligar eletricamente a alimentação do aparelho a eles conectados.

RF1.1 O usuário deverá poder verificar e alterar o estado da conexão através da interface web a qualquer momento.

RF2. Os produtos devem ser capazes de gerar eventos que possam ser aproveitados pelos demais dispositivos da rede a fim de que estes realizem ações de acordo com o evento recebido.

RF3. O dispositivo deverá possuir um nome de maneira a identificá-lo na rede.

RF3.1 O usuário poderá atribuir o nome ao dispositivo.

RF4. Os dispositivos devem estar sincronizados com o horário atual, fixados no fuso GMT-3.

RF5. O usuário poderá programar ações de energização ou desligamento de aparelho atreladas a um horário.

RF6. Os dispositivos devem possuir um modo de configuração, onde disponibilizarão uma interface onde o usuário deve fornecer os dados de sua rede local sem fio para que o dispositivo a ela se conecte.

RF6.1 O usuário poderá retornar o dispositivo a este modo de configuração através de um botão.

RF7. Os dispositivos devem possuir um LED que indique o estado atual da conexão à rede WLAN.

RF7.1 O LED de conexão deve piscar com frequência de 1Hz quando o dispositivo estiver em modo configuração;

RF7.2 O LED deve piscar com frequência de 2Hz quando o dispositivo estiver tentando se conectar à rede WLAN do usuário;

RF7.3 O LED deve ficar aceso ininterruptamente enquanto o dispositivo estiver conectado à rede WLAN do usuário.

RF8. Os dispositivos devem possuir um LED que indique o estado atual da energização do aparelho conectado à sua saída controlada.

RF8.1 O LED de energização deve ficar apagado quando o dispositivo estiver desconectado da rede de energia;

RF8.2 O LED de energização deve ficar aceso quando o dispositivo estiver conectado à rede de energia.

3.2.2. Requisitos não funcionais

RNF1 A solução deve implementar em uma única placa de circuito impresso a possibilidade de se construir tanto o dispositivo tomada quanto interruptor;

RNF2 A solução deve buscar a maior integração possível, favorecendo circuitos integrados onde microcontrolador e controlador de comunicação sem fio se encontrem no mesmo encapsulamento;

RNF3 O microcontrolador escolhido deve suportar uma implementação existente de sistema operacional de tempo real, a fim de facilitar e organizar a implementação do software embarcado.

3.3. Casos de uso

Os casos de uso buscam demonstrar o relacionamento de um agente (usuário ou dispositivo da mesma linha) com o sistema proposto através da descrição de situações onde os requisitos funcionais sejam utilizados.

3.3.1. Primeiro uso do produto

O usuário conecta o produto à rede elétrica de sua casa. O produto então faz anúncio de uma WLAN cujo nome e senha estão disponíveis no manual. O usuário se conecta a esta rede utilizando um computador, *smartphone* ou dispositivo similar capaz de acessar WLANs e exibir páginas web. O usuário abre a interface de configuração através de um navegador acessando um IP específico disponível no manual. Nesta interface constam campos de entrada para o nome da WLAN do usuário, senha da WLAN do usuário e para o nome que se deseja associar ao dispositivo. Ao entrar com as informações, o dispositivo irá tentar se conectar à rede especificada pelo usuário, assumindo o nome que o usuário lhe atribuiu.

3.3.2. Controle remoto

Com ambos usuário e dispositivo conectados à mesma WLAN, o usuário acessa a interface do sistema através de um IP concedido pelo roteador ao dispositivo. A interface mostra as informações de todos os dispositivos conectados, e através de botões permite ligar ou desligar a energização dos aparelhos conectados aos dispositivos.

3.3.3. Programação de tabela horária

Através da interface mencionada anteriormente, o usuário pode programar horários para que os aparelhos conectados aos dispositivos sejam energizados ou desligados.

3.3.4. Programação de comportamento

Através da interface, o usuário pode programar uma dinâmica que relaciona um evento no elemento sensor de um dispositivo com o energizar ou desligar de outros aparelhos conectados a dispositivos da rede.

3.3.5. Retorno ao modo configuração

O dispositivo deverá voltar ao modo de configuração em duas situações: caso não consiga se conectar à rede do usuário na primeira tentativa após a configuração ou através do pressionar de um botão.

4. IMPLEMENTAÇÃO

A implementação abrange o projeto e a realização de um protótipo funcional que busca atender aos requisitos levantados no capítulo anterior, sendo os protótipos desenvolvidos a realização material deste projeto. A implementação buscou fazer uso de tantas tecnologias já existentes quanto possível, de maneira a simplificar o projeto. São explicitadas as decisões entre a utilização de soluções disponíveis e abertas e a utilização de soluções próprias condicionadas a esta aplicação.

O protótipo implementado é otimizado para o uso em bancada de desenvolvimento, a fim de facilitar ciclos de gravação e depuração, servindo como prova de conceito da implementação dos requisitos. Nos capítulos seguintes são discutidos os passos necessários e as pendências a serem resolvidas para tornar o protótipo um produto pronto para as prateleiras de uma loja.

4.1. Escolhas realizadas

4.1.1. Bluetooth versus Wi-fi

A tecnologia de comunicação sem fio escolhida foi a *wi-fi* padrão IEEE 802.11. Os pontos levados em consideração foram os seguintes:

O modelo de rede implementado pelas duas tecnologias: no caso do *bluetooth* os dispositivos conectam-se uns aos outros na topologia ponto-a-ponto, enquanto no caso da *wi-fi* todos os dispositivos se conectam através de roteadores, estes geralmente já disponíveis nas propriedades dos usuários.

Aliado ao modelo de rede está o fator da distância de comunicação que os dispositivos que implementam tais tecnologias conseguem suportar. *Bluetooth* suporta poucos metros de distância entre cada ponto, não suportando uma conexão estável entre pontos instalados em diferentes cômodos de uma residência, por exemplo. Já um dispositivo *wi-fi* consegue se conectar ao modem mesmo com paredes no meio do caminho.

A escolha da *wi-fi* sobre o *bluetooth*, baseando-se nos pontos acima, se deve à intenção de providenciar ao usuário a possibilidade de instalar os produtos em diferentes cômodos de sua propriedade, restrito apenas à potência do sinal *wi-fi* disponível no ambiente.

O *bluetooth* seria mais aplicável a uma rede de dispositivos onde exista um espaçamento constante e pequeno entre os pontos.

A título de conhecimento: a tecnologia *ZigBee* – similar às tecnologias discutidas anteriormente, porém com custo e alcances reduzidos – não foi considerada pois transceptores *ZigBee* não são encontrados em dispositivos de uso pessoal como *smartphones* e computadores, o que acarretaria em um custo final maior para a solução devido à necessidade de um dispositivo que realize a ponte entre a tecnologia *ZigBee* e uma outra (USB, *bluetooth*, *wi-fi*) que o usuário tenha acesso.

4.1.2. Interface web versus aplicativos

Foi escolhida a interface web visando a simplificação do projeto. Uma interface via aplicativo demandaria implementações condicionadas às várias plataformas disponíveis – sistemas operacionais para computadores e *smartphones* – gerando artefatos adicionais para a manutenção ou limitando as maneiras com que o usuário poderia interagir com os produtos.

Com a interface web qualquer dispositivo moderno capaz de navegar na internet se torna uma interface com a rede, diminuindo a complexidade do projeto.

4.1.3. System-on-chip utilizado

Buscando atingir o requisito não-funcional (RNF2) onde o mesmo circuito integrado realize as funções de controlador e de ponte de comunicação sem fio foram eleitos dois circuitos integrados, o CC3200 da Texas Instruments e o ESP8266 da Espressif.

O CC3200 é um *system-on-chip* baseado em um núcleo ARM Cortex-M4, possuindo um controlador dedicado à comunicação sem fio que implementa protocolos da camada física (IEEE 802.11 b/g/n), rede (IP) e transporte (TCP/IP). É de propriedade da Texas Instruments, possuindo um *software development kit* (SDK) robusto e consolidado no mercado, suportado por uma vasta documentação e ferramentas de desenvolvimento. Também possui uma implementação de RTOS exclusiva à Texas Instruments – o TI-RTOS – que é uma alternativa ao SDK convencional que integra os *drivers* do dispositivo aos elementos do sistema operacional. Um kit de desenvolvimento custa U\$30,00, adquirido direto com a empresa fabricante ou através de revendedores.

O ESP8266 é um *system-on-chip* desenvolvido pela empresa Espressif, baseada na China, contendo um núcleo Tensilica L106 Diamond de 32 bits e um controlador dedicado à comunicação sem fio com especificações equivalentes ao

CC3200. O fabricante fornece dois kits de desenvolvimento de software: um voltado para programação assíncrona baseada em *call-backs* e outro baseado em uma adaptação do Free-RTOS 7.5. Sua documentação é escassa, com documentos explicando como preparar o compilador e breves descrições das funções fornecidas e de exemplos básicos. Seu maior atrativo é certamente o custo: uma placa contando com o SoC, conector USB para alimentação e comunicação e todos os pinos expostos através de conectores para *proto-board* custa U\$5,00, preço que a tornou muito atraente para qualquer pessoa que queira desenvolver um projeto com conectividade *wi-fi*. Este preço gerou toda uma comunidade aberta em torno da plataforma, com muita discussão em fóruns técnicos e repositórios contendo projetos de aplicações com código aberto utilizando esse *chip*.

Ambas as plataformas oferecem SDKs que integram o controle de periféricos a uma implementação de RTOS. Baseando-se no custo para se adquirir placas de desenvolvimento foi escolhido o ESP8266 para o desenvolvimento deste projeto, considerando que o conteúdo disponível em fóruns e *blogs* na internet fosse suficiente para cobrir a deficiência em documentação que a plataforma da Espressif apresenta quando comparada àquela da Texas Instruments.

4.1.4. Acesso remoto

Para se possibilitar o acesso remoto à rede de dispositivos – isto é, o cliente poder acessar e controlar os dispositivos fora de sua WLAN, via conectividade 4G ou WLAN de outros locais – duas alternativas foram vislumbradas.

A primeira necessitaria de que o cliente entrasse em contato com sua provedora de internet a fim de conseguir um IP fixo, e após isso o cliente deveria de configurar seu modem a fim de redirecionar as conexões às portas deste IP para o IP dos dispositivos da rede. Isto vai contra a simplicidade estipulada nos requisitos e descrição do projeto, terceirizando ao cliente um trabalho moroso de contato com a provedora de serviço, sujeito a disponibilidade e custos do IP fixo.

A segunda alternativa seria incluir no projeto um servidor, de posse da empresa que fabricaria os produtos deste projeto, onde os produtos se conectariam e trocariam informações. O usuário então poderia também se conectar a este servidor e através de um serviço protegido por autenticação poderia verificar e controlar o estado dos produtos em sua propriedade. Esta seria a alternativa mais elegante, porém resultaria num custo de operação e manutenção de servidores especializados,

além de trazer para o projeto a responsabilidade sobre a segurança dos dados dos clientes. Devido à complexidade relacionada ao projeto do servidor, este não será incluído neste projeto.

Posto isso, a rede de produtos que um usuário possui não pode ser acessada remotamente, sendo necessário que o usuário esteja com o dispositivo de acesso conectado à mesma WLAN que os produtos. Como a rede de dispositivos e qualquer comunicação entre dispositivos está limitada à rede local do usuário, estipula-se que é de responsabilidade do usuário a segurança do acesso e do controle da rede de dispositivos – através das configurações de segurança da WLAN disponíveis no modem de sua casa, por exemplo.

4.2. Tecnologias utilizadas

Aqui é feita uma breve descrição de cada tecnologia utilizada no projeto.

4.2.1. ESP8266

O ESP8266, cujo diagrama de blocos pode ser visto na Figura 2, é um *system-on-chip* desenvolvido pela Espressif para suprir as demandas por desenvolvimento de produtos conectados à internet. Segundo o *datasheet* do dispositivo (ESPRESSIF SYSTEMS, 2018) o SoC possui um microcontrolador *Tensilica L106 Diamond series* de 32 bits integrado aos periféricos necessários para se realizar a comunicação sem fio. Ele não possui memória Flash integrada, sendo necessária uma externa conectada a uma interface SPI para armazenar o programa a ser executado.

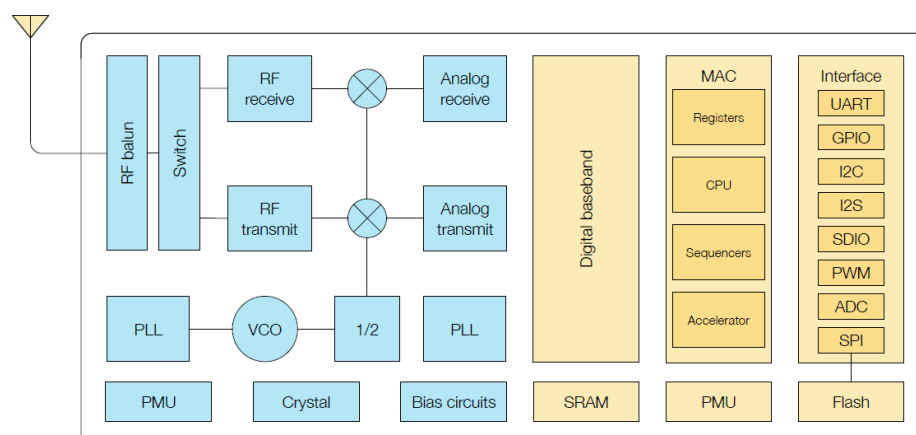


Figura 2: Diagrama em blocos do ESP8266
Fonte: ESP8266 System Description (ESPRESSIF SYSTEMS, 2018)

O ESP8266 possui áreas de memória RAM dedicadas para a execução e armazenamento de código e de dados, sendo 32 KiB¹ de RAM para instruções, 32 KiB para cache e 80 KiB para dados. Ressalta-se que o valor de 80 KiB de RAM é nominal, o uso do SDK do fabricante e das pilhas de comunicação sem fio integradas já toma parte deste montante.

O SoC pode ser utilizado como uma ponte de comunicação serial para *wi-fi* para um microcontrolador externo (utilizando um arquivo binário de código fornecido pela fabricante) ou, segundo o guia de introdução do dispositivo (ESPRESSIF SYSTEMS, 2018, p. 2), pode ser programado como um microcontrolador comum utilizando um dos SDKs fornecidos pelo fabricante: o *Non-OS ESP SDK* e o *ESP RTOS SDK*.

O Non-OS ESP SDK é composto de funções voltadas à programação orientada a eventos, onde o programador escreve funções que servem de reações a eventos como recebimento de conexões, recebimento de mensagens ou passagens de tempo.

O ESP RTOS SDK é voltado à programação utilizando um sistema operacional – neste caso o Free-RTOS 7.5 – onde o programador escreve tarefas, tendo a sua disposição ferramentas de controle e de comunicação entre tarefas. Este foi o SDK selecionado para o desenvolvimento deste projeto.

Além dos SDKs anteriormente citados, a comunidade de desenvolvedores que utilizam a plataforma desenvolveu uma série de adaptações resultando em novos SDKs de código aberto. Dentre eles têm destaque: um SDK baseado na plataforma Arduino, possibilitando a programação em C++ com chamadas e ferramentas do ecossistema Arduino; um SDK chamado ESP OPEN RTOS, que se propõe a ser uma alternativa melhor que o ESP RTOS SDK, baseado no Free-RTOS 10 com um ciclo de atualizações mais rápido; um SDK chamado NodeMCU, baseado na linguagem de programação Node.js que tem como objetivo facilitar e simplificar a criação de dispositivos conectados.

O SoC é encontrado no mercado em módulos contendo a Flash SPI e a antena, e ao longo do desenvolvimento deste projeto foi utilizado o módulo ESP-12E.

¹ A unidade kibibyte (KiB) é uma unidade de informação digital que representa 1024 (2^{10}) bytes. Os prefixos kibi, mibi e gibi, representando respectivamente 2^{10} , 2^{20} e 2^{30} unidades, fazem parte da normal IEC 80000/13.

Segundo seu manual de referência (AI-THINKER, 2018), o módulo ESP-12E, visto na Figura 3, integra o SoC ESP8266 com uma memória Flash SPI de 4 MiB, uma antena e um LED de status. Disponibiliza todos os pinos GPIO do SoC em uma pequena placa, sendo ideal para a montagem de protótipos ou produtos. Foi utilizado na montagem da placa projetada para os produtos.



Figura 3 - Módulo ESP12-E
Fonte: AI-Thinker

4.2.2. Free-RTOS

O Free-RTOS é um sistema operacional de tempo real que fornece ao programador que o utiliza ferramentas para criar um sistema com tempo controlado de resposta a eventos. As ferramentas são compostas de tarefas – funções criadas para se cumprir um requisito – e estruturas para a comunicação e controle de tarefas. As tarefas podem ter sua execução interrompida por outras tarefas de maior prioridade ou podem ser bloqueadas enquanto esperam por um recurso (físico ou virtual) estar disponível.

As ferramentas utilizadas do FreeRTOS para a execução deste projeto foram as seguintes:

- Tarefas: Foram criadas tarefas para cada funcionalidade da aplicação embarcada a fim de se modularizar o código;
- Filas: Foram utilizadas filas de dados a fim de se organizar a transferência de dados e requisições entre tarefas;
- Mutex: Foram utilizados mutexes para proteger recursos, como a saída serial de depuração e o sistema de arquivos, a fim de que somente uma tarefa possa acessar um dado recurso de cada vez, e que consiga completar a operação iniciada sem que outra tarefa comece a utilizar o

recurso – o que acarretaria em corrupção de dados ou perda de controle do sistema.

4.2.3. Protocolos TCP/UDP

O ESP8266 possui implementações internas de pilhas TCP e UDP, abstraindo qualquer necessidade de controle do programador através das SDKs disponíveis.

O protocolo TCP (*transmission control protocol*) permite que um dispositivo cliente se conecte a um dispositivo servidor a fim de trocar pacotes de dados através de uma sessão, onde a ordem e a integridade dos pacotes trocados são asseguradas. É utilizado no projeto principalmente pela implementação do servidor HTTP, responsável por servir as páginas da interface web quando requisitadas por um navegador (cliente).

O protocolo UDP (*user datagram protocol*) é semelhante ao protocolo TCP, porém sem a estrutura de uma sessão de comunicação. Não há procedimentos para o estabelecimento de conexão entre os dois dispositivos, não existindo garantia de ordem ou recepção dos pacotes enviados, o que torna este protocolo menos confiável e mais célere com relação ao TCP. Neste projeto o protocolo UDP foi utilizado para a comunicação entre dispositivos na WLAN, baseado nas premissas de que as mensagens a serem trocadas são pequenas, independentes (não existindo mensagens compostas por mais de um pacote) e de que os dispositivos se encontram próximos uns dos outros na rede lógica – existindo poucos dispositivos no caminho que o pacote percorre entre os envolvidos, um único roteador por exemplo. O protocolo UDP também é utilizado pelo sistema indiretamente pelo protocolo SNTP (subseção 4.2.6), onde o dispositivo atua como cliente requisitando a um servidor a *timestamp* atual a fim de realizar as funções relacionadas à temporização.

4.2.4. Protocolo HTTP

O HTTP (*hyper text transfer protocol*) é um protocolo de camada de aplicação utilizado para a transferência de documentos e mídia entre um cliente e um servidor. É um protocolo sem memória de estado, onde o servidor não guarda nenhuma informação sobre a ordem ou dados trocados nas requisições.

Este protocolo é utilizado pelo sistema para servir a navegadores clientes as páginas web que compõem a interface, embasado sobre o protocolo TCP.

4.2.5. Linguagens de programação de páginas web

Páginas web são documentos de texto puro contendo códigos a serem interpretados pelo navegador que as requisitou. Uma página web moderna é composta por três linguagens de programação: o HTML (*HyperText Markup Language*), o Javascript e o CSS (*Cascading Style Sheets*).

O HTML é uma linguagem de descrição de estrutura responsável pela composição de uma página web. Ele descreve a ordem dos elementos – texto, imagens, links - que compõem uma página através de etiquetas (*tags*). O documento HTML é a base de uma página web, e é nele que as outras duas linguagens – Javascript e CSS - são inseridos.

Javascript é uma linguagem interpretada pelo navegador capaz de manipular a estrutura do código HTML de que uma página é composta como resposta a eventos gerados por entradas do usuário e do sistema. É utilizada pelo sistema a fim de tornar a interface dinâmica, atualizando a página periodicamente requisitando ao servidor (dispositivos) novas informações a respeito do estado dos dispositivos na rede.

CSS é uma linguagem de descrição de estilo, atuando sobre o código HTML da página. É a responsável por alterar as cores, tamanhos, espaçamentos e estilos em geral da página web para dar um tom mais moderno e agradável, além de ser a responsável por alterar a disposição dos elementos quando a página é acessada por um computador – com telas grandes – ou por um smartphone – com a tela pequena. O conceito de uma página web que se adequa ao dispositivo no qual é acessada se chama projeto responsivo (do inglês *Responsive Design*), e foi implementado neste projeto.

4.2.6. SNTP

Simple Network Timing Protocol é um protocolo de aplicação construído sobre o protocolo UDP, utilizado para requisitar de um servidor remoto uma *timestamp* atualizada. Foi utilizada uma implementação terceirizada do serviço através da biblioteca LwIP (*a lightweight TCP/IP stack*), onde as transações são realizadas através de chamadas onde o programador fornece o IP do servidor que deseja requisitar *timestamp* e onde o programador requisita a *timestamp*.

4.2.7. Rest API

Representation State Transfer é um tipo de serviço onde requisições de dados ou ações podem ser realizadas através do protocolo HTTP com um tamanho reduzido de dados. Voltado aos novos dispositivos conectados à internet, onde geralmente não há memória disponível para se apresentar páginas web inteiras, possibilita ao cliente requisitar um artefato específico de um servidor sem toda a estrutura de página web que normalmente o acompanharia. Por exemplo, um dispositivo pode requisitar ao servidor qual a previsão do tempo para a cidade na qual está inserido – sem o HTML associado à página web que um navegador exibiria.

Para tornar este projeto condizente com esta tecnologia, uma API foi criada como <ip do dispositivo>/api/on e <ip do dispositivo>/api/off, que liga ou desliga os aparelhos conectados aos dispositivos.

4.2.8. SPI Flash File System (SPIFFS)

O módulo ESP-12E utilizado no projeto possui uma Flash SPI de 4MiB, porém o processador pode usar somente o primeiro 1MiB como memória de programa. O restante pode ser aproveitado como memória não volátil de dados, através de instruções de escrita e leitura na memória SPI em endereços acima de 0x0010 0000.

A fim de se colocar uma camada de abstração por cima do acesso à memória flash foi utilizada uma biblioteca terceirizada de sistema de arquivos chamada SPIFFS. Esta biblioteca possibilita o manuseio de um setor de memória flash na forma de arquivos com chamadas estilo *posix* – *open*, *close*, *read*, *write*, *seek* entre outros – além de checar consistência nos momentos de inicialização, abertura e fechamento de arquivos, auxiliando na prevenção e tratamento de dados corrompidos

A gravação de dados em memória não volátil foi utilizada no projeto a fim de se salvar informações como dados da WLAN do usuário, último estado da energização do aparelho conectado, tabela horária e tabela de programação da rede.

4.3. Projeto de hardware

O projeto de hardware contemplou o diagrama esquemático e a escolha de componentes do protótipo, o *layout* da placa de circuito impresso e o processo de solda dos componentes no protótipo.

4.3.1. Esquemático e componentes

O hardware do protótipo é compreendido em quatro grupos de circuitaria: o retificador, o conversor DC-DC, o módulo processador e os drivers/ condicionadores de atuadores e sensores.

4.3.1.1. Retificador

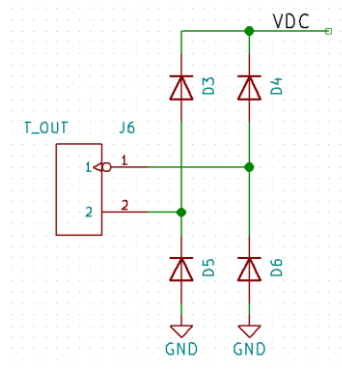


Figura 4: Retificador
Fonte: Autoria própria

A Figura 4 mostra o retificador utilizado para o projeto. Foi escolhida a topologia ponte completa a diodos – D3, D4, D5 e D6 como diodos de modelo 1N4007 – a fim de suportar como entrada de tensão tanto a saída de um transformador de tensão (de primário 127V e secundário 9V ou 12V, por exemplo) como de uma fonte DC de 9V a 12V (com propósito de facilitar a prototipagem).

Outra alternativa considerada teria sido ao invés de um transformador utilizar um divisor capacitivo para abaixar os 127V RMS da rede para um nível menor suportável pelos conversores DC-DC. Essa solução reduziria o custo e o tamanho físico do projeto, porém não proveria o isolamento elétrico do protótipo, diminuindo a segurança do protótipo.

4.3.1.2. Conversor DC-DC

O protótipo necessita de dois níveis de tensão DC: 3,3V para o módulo ESP-12E com capacidade de corrente de até 400mA e 5,0V para o relé e sensor infravermelho, com capacidade de corrente de até 100mA. A entrada do conversor DC-DC é estipulada de 7V a 10V DC.

Para o circuito de 3,3V foi selecionado um conversor tipo *buck* chaveado da Texas Instruments, o LM2596, que possui valor de tensão de saída configurável através de uma realimentação por divisor de tensão resistivo. Mais especificamente,

foi selecionado o LM2596-3.3, que é a variante com o circuito de realimentação integrado já condicionado para gerar a saída de 3,3V, eliminando componentes da lista de materiais e garantindo maior precisão ao valor de saída. O circuito montado na placa – ilustrado pela Figura 5 – segue a aplicação típica explícita no *datasheet* (TEXAS INSTRUMENTS, 2018) do dispositivo.

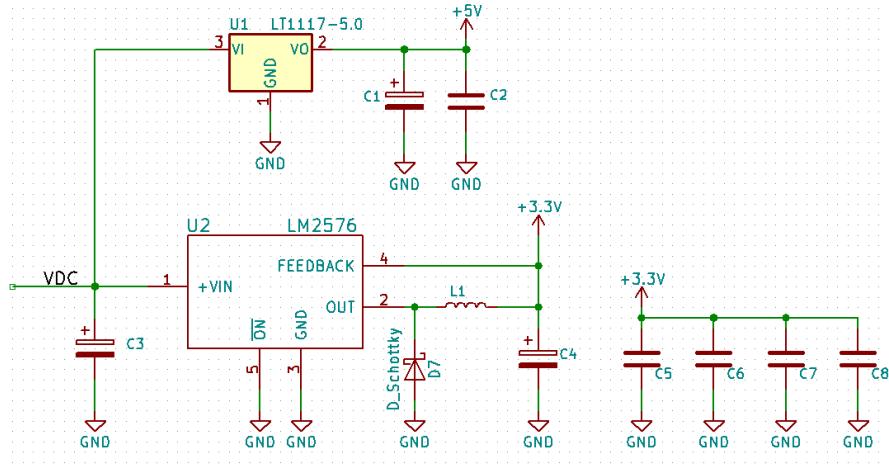


Figura 5: Conversores DC-DC do protótipo
Fonte: autoria própria

O diodo *schottky* utilizado (D7) foi o 1N5819, com corrente média máxima de 1A e pico de tensão reversa máximo de 40V, características que cobrem com folga o funcionamento normal do protótipo. Os capacitores C3 e C4, de tântalo com valor 470uF, eliminam o *ripple* de frequências baixas, enquanto os capacitores C5, C6, C7 e C8, de cerâmica, eliminam o *ripple* de altas frequências, buscando contrabalancear a ESR (*equivalente series resistance*) do capacitor de tântalo. São posicionados próximos às alimentações do módulo ESP12-E e dos demais blocos da placa.

O circuito integrado LT1117 – conversor de tensão linear – foi escolhido para a regulação da alimentação de 5V, mais especificamente sua versão condicionada a uma saída fixa de 5V, o LT1117-5.0. A série 1117 é fornecida por diversos fabricantes com pouca variação de especificação entre seus dispositivos. Em sua versão de saída fixa não necessita de nenhum componente adicional, sendo C1 um capacitor de tântalo de valor 220uF para retirar o *ripple* de baixas frequências e C2 um capacitor de cerâmica de valor 100nF para retirar o *ripple* de altas frequências.

4.3.1.3. Módulo controlador

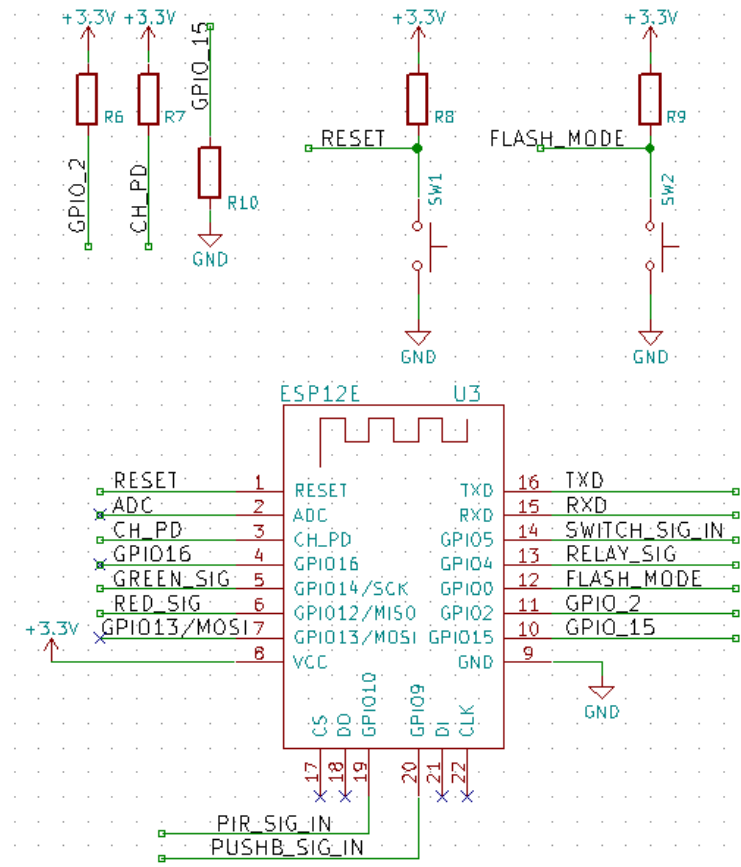


Figura 6: Módulo controlador
Fonte: autoria própria

Para fins de projeto de hardware o módulo ESP-12E foi tratado como um componente. A Figura 6 ilustra os componentes utilizados na configuração do módulo. Todos os resistores desta parte do circuito (R6, R7, R8, R9 e R10) têm valor 10kΩ. Os resistores servem para assegurar os níveis descritos pelo Quadro 1 para o correto inicializar do módulo após um evento de *reset*. O botão SW2, através da rota FLASH_MODE, permite que no momento de inicialização se decida se deseja executar o programa salvo na Flash SPI ou se deseja carregar um novo programa.

Quadro 1: Modos de inicialização do ESP-12E

| Modo de inicialização | GPIO0 | GPIO2 | GPIO15 |
|--------------------------------|-------|-------|--------|
| Download de programa pela UART | L | H | L |
| Ler programa da Flash SPI | H | H | L |

Fonte: AI-Thinker, 2018, p. 6.

4.3.1.4. Drivers e condicionadores de sinal

Ilustrada pela Figura 7, este bloco engloba os circuitos que realizam a interface entre os pinos do módulo ESP-12E e os demais periféricos.

Para os LEDs e relé, um simples circuito driver baseado em um transistor tipo NPN é utilizado – R4, R13 e R14 são resistores de $1k\Omega$ e R5, R15 e R16 são resistores $10k\Omega$. O relé é acionado com 5V, e o diodo de roda livre (D2) é um 1N4007. Para limitar a corrente nos LEDs, resistores de 330Ω são montados em série (R11 e R12).

Para as entradas SWITCH (J3) e PB (J1), que servem para conectar uma chave estacionária e um *push-button* respectivamente, são montados resistores *pull-up* de $10k\Omega$ (R2 e R3).

Para a entrada do sensor PIR, inicialmente foi projetado um abaixador de tensão formado pelo resistor R1 e o zener D1, porém foi verificado empiricamente que o sinal de saída do sensor PIR possuía nível alto em 3V, então o zener D1 não é soldado na placa e o resistor R1 é soldado com valor 0Ω .

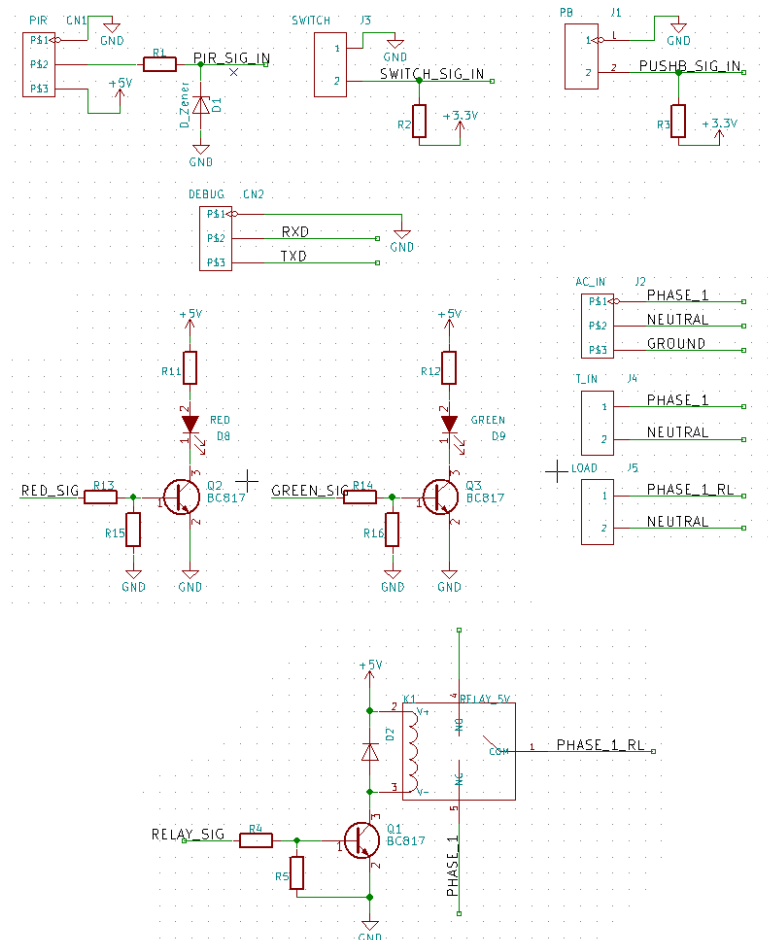


Figura 7: Circuitos *drivers*, condicionadores de sinal e conectores da placa
Fonte: autoria própria

4.3.1.5. Módulo PIR

Para a montagem do produto “tomada inteligente” é utilizado um sensor PIR – *passive infrared sensor* – a fim de se detectar presença de pessoas. O sensor adquirido para a montagem dos protótipos não possuía identificação de modelo ou fabricante, tornando impossível achar um manual referente à parte em questão. Para se trabalhar com o componente foi utilizado um guia de referência da empresa fabricante de módulos eletrônicos Adafruit, identificando-se similaridades entre o sensor em mãos e aqueles descritos no guia. O módulo, visto na Figura 8, é constituído por uma lente de Fresnel, um conjunto de elementos sensores e circuitos condicionadores de sinal que transformam o sinal elétrico do conjunto de sensores em um único sinal de saída. O sensor possui três pinos, VCC, GND e SINAL, e é conectado ao protótipo através do conector CN1 (visto na Figura 7).



Figura 8: Módulo sensor PIR
Fonte: PIR Motion Sensor (Adafruit, 2017, p. 3)

Segundo Adafruit (2017), o sensor em si possui dois elementos sensores de luz infravermelha, e quando os elementos do ambiente se encontram em repouso ambos os sensores recebem a luz infravermelha com intensidade igual. Quando um elemento emissor de luz infravermelha se move, por um instante de tempo um dos sensores irá receber mais luz do que a outra, e a duração deste instante é utilizada para disparar o sensor, conforme ilustrado pela Figura 9.

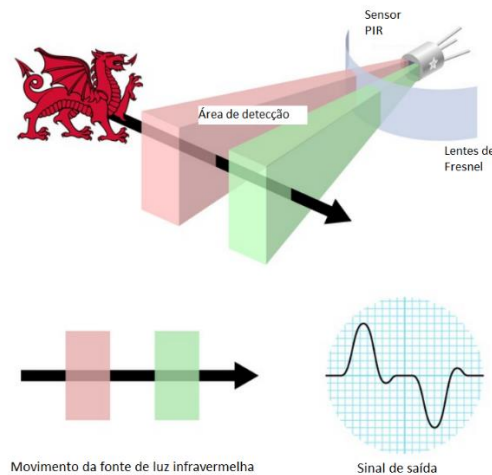


Figura 9: Esboço de detecção de movimento por parte do sensor PIR
Fonte: adaptação de PIR Motion Sensor (Adafruit, 2017, p. 6)

Quando disparado, o sensor emite um pulso em nível alto (3,3V) por um tempo determinado na faixa de segundos. Este tempo pode ser calibrado a partir de um resistor ajustável no módulo PIR, bem como a duração do movimento para que o sensor dispare.

4.3.2. Layout da placa de circuito impresso

A placa de circuito impresso foi projetada utilizando o programa KiCAD. Ele foi escolhido por ser completamente gratuito, possuir editor de esquemático e de placa de circuito, possuir bibliotecas com símbolos de componentes e *footprints* para PCBs e ser capaz gerar lista de materiais, arquivos para fabricação e visualização 3D.

Durante o projeto da placa foram priorizados os seguintes aspectos:

- Utilizar tanto quanto possível do plano *bottom* para ligação com o GND, implementando um plano de terra;
- Utilizar o espaço sobrando do plano *top* para a ligação com 3,3V;
- Utilizar planos diferentes para rotear as ligações de 127V – fase e neutro;
- Optar sempre por vias até o plano GND ao invés de realizar ligações no plano *top*.

Como tutorial de uso do programa KiCAD foi utilizada uma série de artigos intitulados *Creating a PCB in everything: KiCAD* (BENCHOFF, 2016). Como resultado do projeto de hardware obteve-se a placa vista nas imagens a seguir: vista superior (*top*) na Figura 10, vista inferior (*bottom*) na Figura 11, e uma simulação de vista 3D da placa na Figura 12.

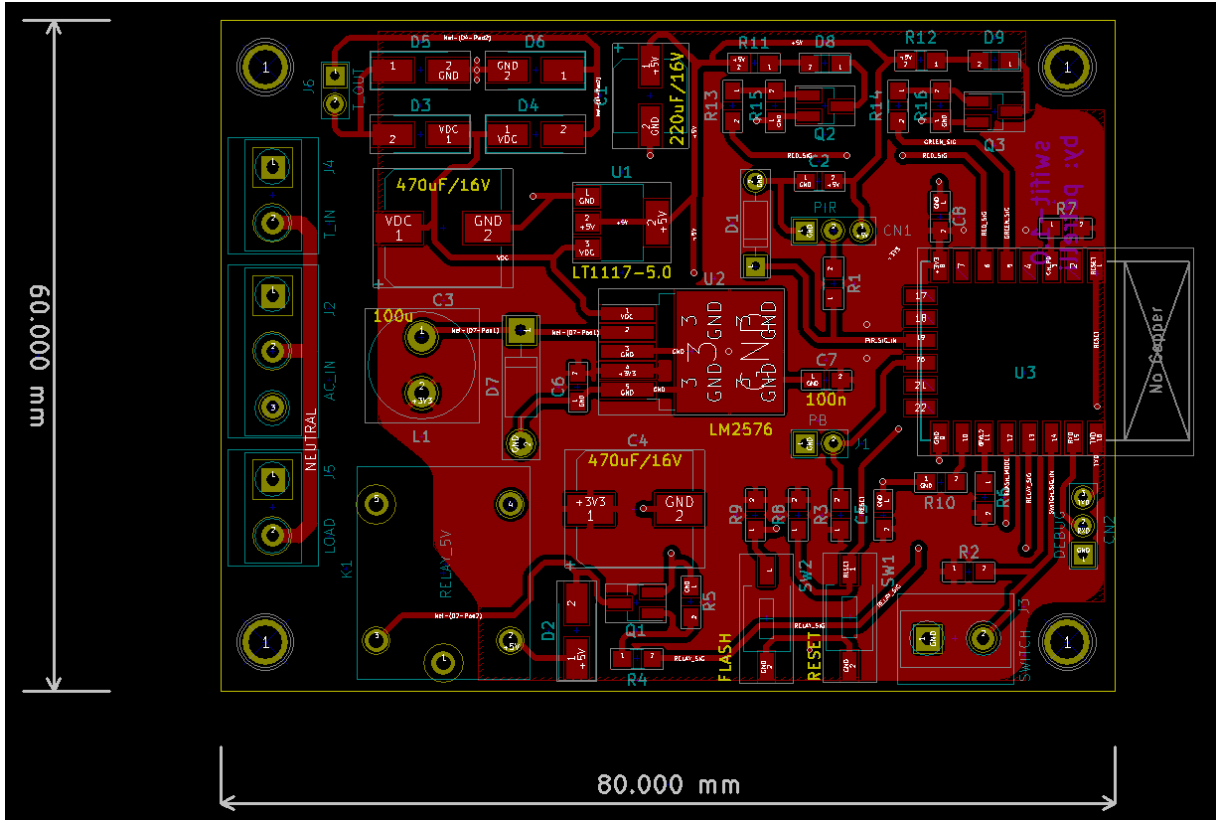


Figura 10: Layout da PCB, vista TOP
Fonte: autoria própria

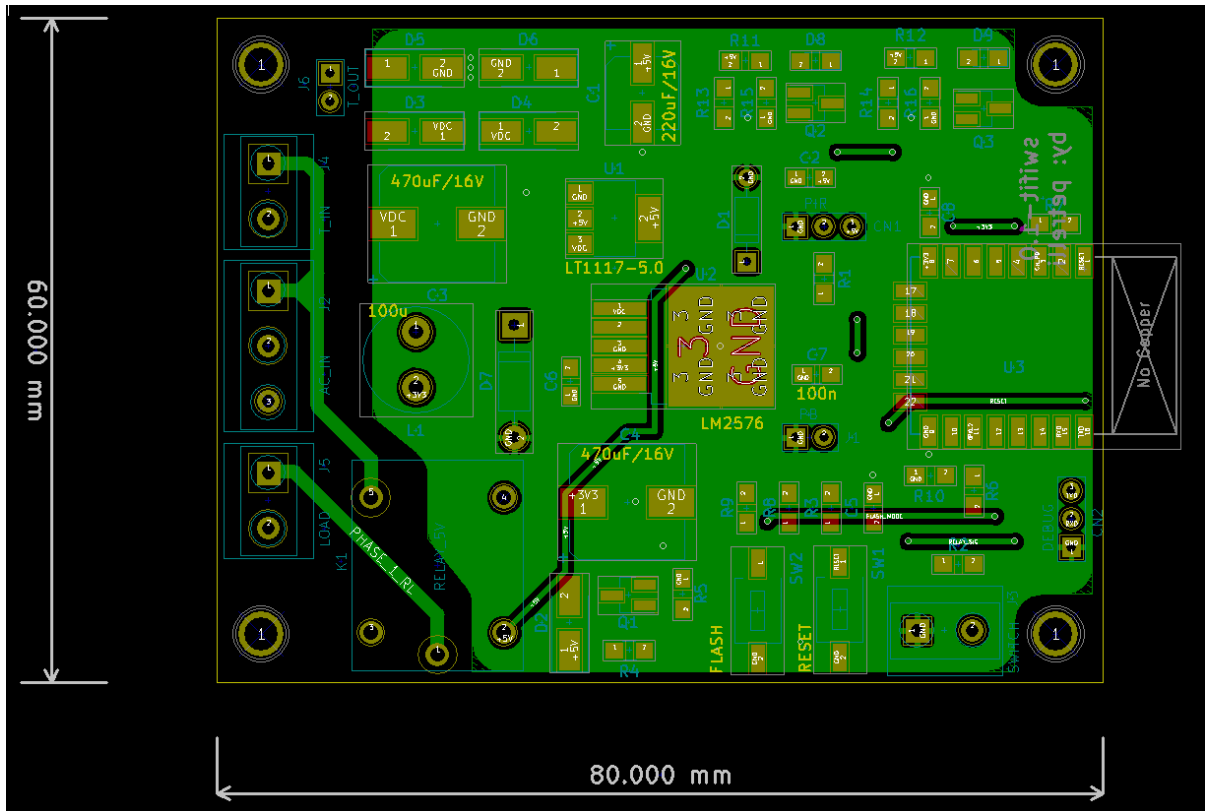


Figura 11: Layout da PCB, vista BOTTOM
Fonte: autoria própria

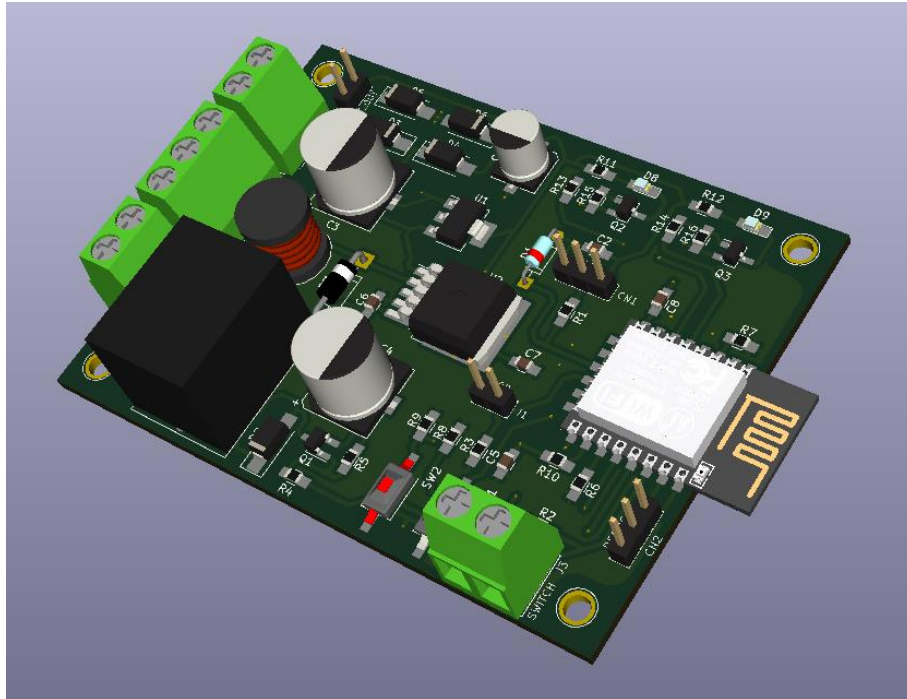


Figura 12: Modelo 3D da PCB
Fonte: autoria própria

Para a fabricação eletrônica foi escolhido o serviço Fusion da Seedstudio, uma empresa chinesa.

4.4. Projeto de comportamento e de rede

O projeto de rede abrange o comportamento geral dos dispositivos, especificando as atividades de cada dispositivo na WLAN do usuário em face aos eventos possíveis.

4.4.1. Descrição comum

A especificação é de que o sistema seja ciente de todos os dispositivos conectados à WLAN do usuário, e que o usuário possa, através de uma única interface, controlar e verificar todos os dispositivos. Como todos os dispositivos estão conectados realmente somente a um ponto de acesso sem fio, uma rede lógica virtual entre os dispositivos é criada.

Esta rede lógica é modelada seguindo a forma de estrela, onde um dispositivo assume o papel de mestre e os demais dispositivos que venham a se conectar à rede assumem o papel de escravos. É estabelecida uma ligação virtual entre os dispositivos escravos e o dispositivo mestre, e nenhuma ligação é realizada entre dispositivos escravos. A Figura 13 ilustra uma rede de três produtos conectados à WLAN do usuário através de um roteador.

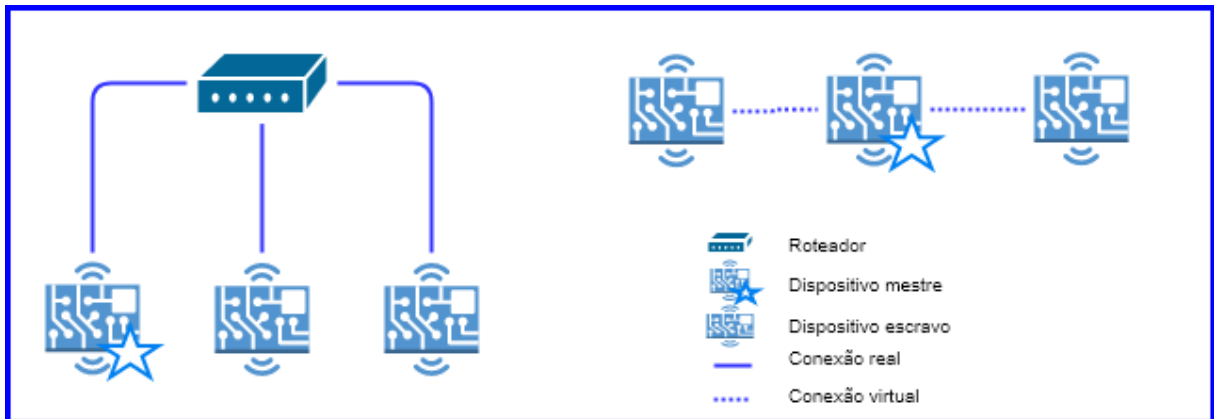


Figura 13: Rede real e rede virtual
Fonte: autoria própria

A Figura 14 ilustra o diagrama de atividades realizadas pelo dispositivo após ser energizado. Ao se inicializar o produto pela primeira vez (consumidor tira o produto da caixa e o conecta à energia elétrica), o produto irá criar uma rede wi-fi com nome e senha pré-definidos de conhecimento do usuário (através do manual). Ao se conectar a essa rede, o usuário acessa o IP do dispositivo através do navegador web, o que disparará o servidor HTTP que irá servir o dispositivo com a página de configuração. O dispositivo irá salvar as informações recebidas e resetar, tentando se conectar a WLAN quando inicializar novamente.

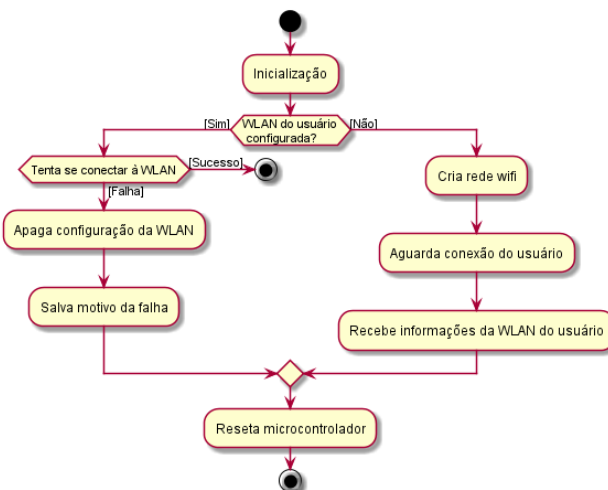


Figura 14: Atividades na primeira inicialização do produto
Fonte: autoria própria

Caso o dispositivo falhe na tentativa de se conectar à WLAN, ele irá reiniciar e retornar ao modo de configuração, fornecendo informações sobre o problema encontrado na página web.

Caso bem-sucedido o dispositivo aguarda uma mensagem UDP de *broadcast* que deve ser enviada pelo dispositivo mestre. Caso seja o primeiro

dispositivo a se conectar a esta rede esta mensagem não chegará, e o dispositivo assumirá o posto de mestre. Caso contrário o produto receberá o *broadcast* do mestre, do qual é possível extrair o IP do mestre, e se tornará escravo. A partir deste momento o novo escravo irá responder a qualquer *broadcast* recebida com uma mensagem de estado, contendo informações sobre o tipo de produto e estado dos seus periféricos, indicando sua presença na rede virtual. O diagrama de atividades da Figura 15 resume as atividades de um dispositivo ao se conectar à rede do usuário.

O tempo de espera pela mensagem *broadcast* é aleatório, o que virtualmente impede que dois dispositivos que se conectem à rede ao mesmo tempo (caso de uma queda na WLAN e seu posterior retorno) se tornem mestres. Presumindo uma aleatoriedade real, dois dispositivos nunca esperarão o mesmo tempo para receber a mensagem, sendo assim um deles estourará o tempo antes e se tornará mestre, enviando a mensagem de *broadcast* e tornando o outro dispositivo escravo.

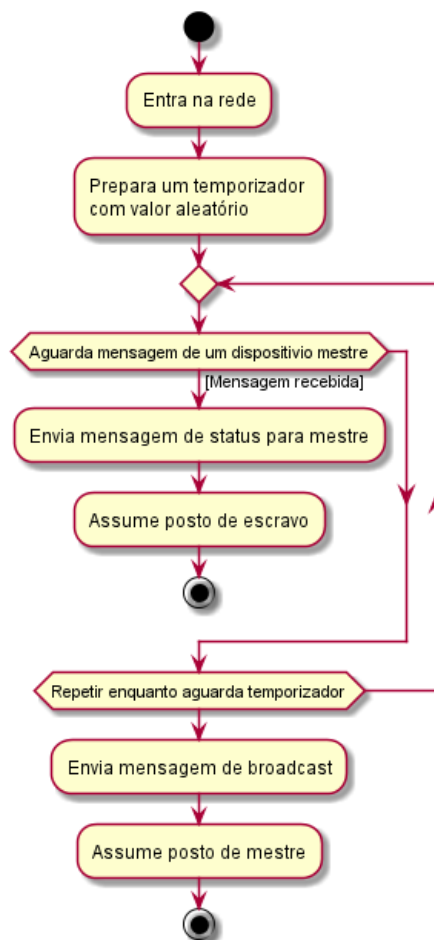


Figura 15: Atividades ao se conectar à uma WLAN
Fonte: autoria própria

4.4.2. Descrição do dispositivo mestre

O dispositivo mestre é encarregado de: repassar as informações recebidas do usuário ou de um escravo para os demais escravos da rede; acumular as informações de todos os dispositivos da rede e servir a interface para o usuário, escutando na porta TCP 80 por uma conexão.

A cada um segundo o dispositivo mestre envia uma mensagem de *broadcast*, indicando sua presença na rede. Ao receber a mensagem *broadcast*, os dispositivos escravos irão enviar uma mensagem de estado para o mestre, que irá atualizar seus dados conforme o recebido. Colisões de pacotes UDP foram desconsideradas no projeto devido à aleatoriedade proporcionada pela transição dos dados pela camada física, pois o tempo entre o software emitir uma mensagem UDP e o destinatário receber tal mensagem fosse aleatório – na faixa de milissegundos – devido à conexão sem fio. A perda de uma única mensagem de estado isoladamente não tem impacto na rede, visto que para que o mestre considere um escravo como desconectado é necessário que este escravo não responda às mensagens de *broadcast* por diversas vezes seguidas. Detalhes da implementação do comportamento de um dispositivo como mestre ou escravo se encontram na seção de implementação de software, Subseção 4.5.4.3.6.

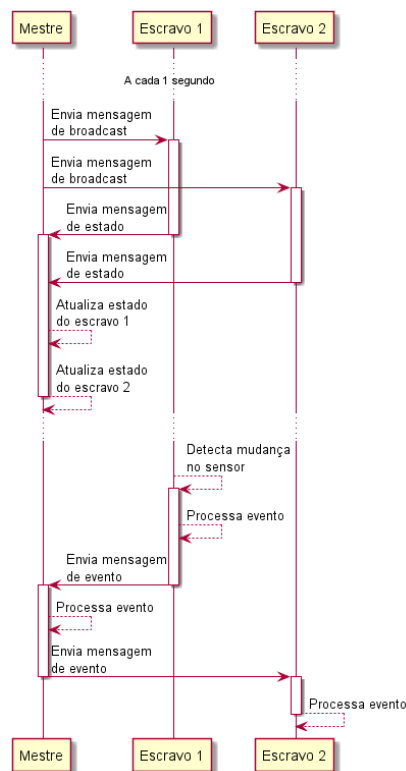


Figura 16: Diagrama de sequência de broadcast e tratamento de eventos
Fonte: autoria própria

O dispositivo mestre também é responsável por enviar aos dispositivos escravos qualquer mensagem de evento recebida – por exemplo quando o sensor PIR de um escravo é disparado.

O diagrama de sequência da Figura 16 ilustra as sequências de *broadcast* e de evento em uma rede com três elementos.

O dispositivo mestre também é responsável por realizar a interface com o usuário. Quando o usuário requisita a interface ou emite ações de acionamento ou reprogramação - de tabela horária ou tabela de respostas aos eventos – é o dispositivo mestre quem deve fazer os repasses para que os dados cheguem aos seus destinatários. O diagrama da Figura 17 ilustra as interações de um usuário com uma rede de dois dispositivos.

Foi verificado empiricamente que em uma rede *wi-fi* – gerada por um roteador D-Link DIR-826L (D-LINK, 2013) – com uma dúzia de dispositivos conectados (celulares, computadores e três protótipos do projeto) o tempo de troca de mensagens permaneceu na faixa de décimos de segundo. Este tempo pode variar de acordo com o número de dispositivos conectados à rede, com a capacidade de processamento do dispositivo roteador e com a qualidade do sinal da rede.

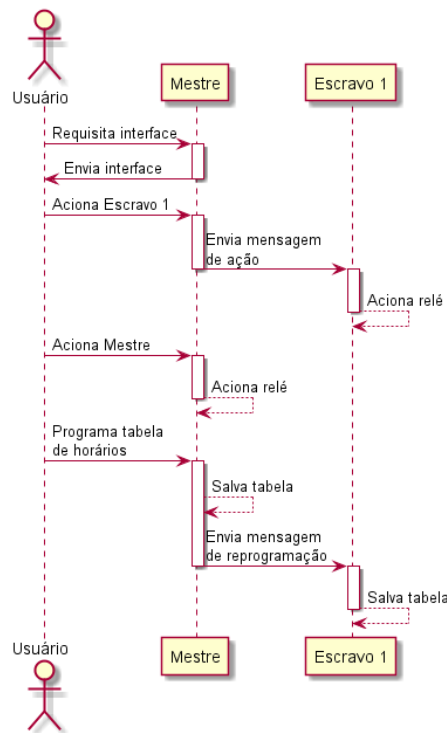


Figura 17: Diagrama de sequências de interações do usuário
Fonte: autoria própria

4.4.3. Descrição dos dispositivos escravos

O dispositivo escravo é responsável por: emitir mensagens de status ao mestre quando receber *broadcasts*, emitir mensagens de eventos gerados em seu sensor, mudar o estado de seu atuador quando receber mensagens de ação e de atualizar suas tabelas temporal e de resposta a eventos quando mensagens de reprogramação forem recebidas.

Os dispositivos escravos devem também manter um temporizador a fim de detectar a saída do dispositivo mestre na rede. Este temporizador é resetado a cada recebimento de mensagens do mestre – qualquer tipo de mensagem rearmar o temporizar com seu tempo inicial. Como o tempo inicial deste temporizador é maior que o período das mensagens de *broadcast* do mestre, o mesmo só há de estourar caso o dispositivo mestre saia da rede. Quando o temporizador estoura, as atividades da Figura 15 são realizadas – o que previne que dois dispositivos assumam a posição de mestre na queda do mestre anterior.

4.4.4. Protocolo de comunicação da rede

O protocolo de comunicação desenvolvido aqui tem como objetivo facilitar a troca de dados entre os dispositivos com o mínimo necessário de processamento e formatação de mensagens. A mensagem é dividida em três segmentos: tipo (TYPE), que enumera o tipo da mensagem; tamanho do campo dados (DCOUNT), que contém o tamanho do campo seguinte; e dados (DATA), que contém os dados que a mensagem busca transmitir, caso aplicável. O Quadro 2 ilustra o formato de uma mensagem do protocolo, com a denominação do campo seguida do tamanho do campo, em bytes, entre parêntesis.

Quadro 2: Datagrama de uma mensagem do protocolo

| | | |
|----------|------------|--------------|
| TYPE (1) | DCOUNT (4) | DATA (0-246) |
|----------|------------|--------------|

Fonte: autoria própria

Como o protocolo é construído sobre o protocolo UDP o IP do remetente pode ser retirado da mensagem recebida. O Quadro 3 resume os tipos de mensagem projetados, mensagens estas discutidas nas seções a seguir.

Quadro 3: Resumo das mensagens da rede

| Tipo de mensagem | ID | Descrição |
|--|----|---|
| <i>Broadcast</i> | 0 | Informar presença de dispositivo mestre na rede |
| <i>Status</i> | 1 | Informar presença de dispositivo escravo na rede |
| <i>Evento</i> | 2 | Informar mudança de estado do sensor de um escravo |
| <i>Ação</i> | 3 | Alterar o estado do atuador de um escravo |
| <i>Programação de tabela horária</i> | 4 | Alterar a programação da tabela horária de um escravo |
| <i>Programação de tabela de regras</i> | 5 | Alterar a programação da tabela de regras de um escravo |

Fonte: autoria própria

4.4.4.1. Mensagem de *broadcast*

Mensagens de *broadcast* têm tipo igual a zero e têm como finalidade informar aos dispositivos que existe um mestre na rede. O IP do mestre é conhecido a partir do remetente da mensagem UDP. O campo de dados desta mensagem é vazio.

4.4.4.2. Mensagem de *status*

A mensagem de *status* é enviada por escravos como resposta às mensagens de *broadcast* recebidas. Seu tipo é igual a um e seu campo de dados contém a cópia de uma estrutura contendo informações a respeito do dispositivo que envia a mensagem.

4.4.4.3. Mensagem de evento

A mensagem de evento é enviada quando um evento é gerado pelo sensor do dispositivo em questão. Seu tipo é igual a dois e seu campo de dados contém a cópia de uma estrutura que agrega o identificador do dispositivo que gerou o evento e o valor do evento em si – se o elemento sensor foi disparado ou retornou ao repouso.

4.4.4.4. Mensagem de ação

A mensagem de ação é enviada pelo mestre a um escravo quando o usuário, através da interface web, dispara um comando para um dispositivo que não o mestre. Seu tipo é igual a três e seu campo de dados contém o valor que o atuador do dispositivo alvo deve assumir.

4.4.4.5. Mensagem de programação de tabela horária

A mensagem de programação de tabela horária é enviada pelo mestre a um escravo quando o usuário, através da interface web, dispara uma reprogramação de tabela horária de um dispositivo que não o mestre. Seu tipo é igual a quatro e seu campo de dados contém a cópia de uma estrutura que representa a nova tabela que o dispositivo alvo deve programar e seguir.

4.4.4.6. Mensagem de programação de tabela de regras

A mensagem de programação de tabela de regras é enviada pelo mestre a todos os escravos quando o usuário, através da interface web, dispara uma reprogramação da tabela de resposta a eventos do sistema. Seu tipo é igual a cinco e seu campo de dados contém uma cópia de uma estrutura que representa a nova tabela de relações entre eventos e ações que os dispositivos devem reprogramar e seguir.

4.5. Projeto de software

4.5.1. Considerações gerais

O software embarcado é todo o código fonte que um microcontrolador deve executar para desempenhar as ações a fim de providenciar as funcionalidades específicas a cada produto. É separado em conjuntos chamados de “camadas” a fim de proporcionar ao projeto as características descritas a seguir.

4.5.1.1. Desacoplamento da plataforma física

Tornar o quanto for possível do código desacoplado da plataforma física que irá executá-lo. Com isto o projeto conseguirá maior flexibilidade e conseqüentemente independência de flutuações no mercado de semicondutores, não dependendo da disponibilidade de um único microcontrolador e fornecedor.

4.5.1.2. Desacoplamento entre módulos

Tornar trechos de código com funcionalidades distintas independentes entre si. Minimizar o referenciamento de código, e documentar detalhadamente a árvore de referências de um dado conjunto de código.

4.5.1.3. Portabilidade

A partir do desacoplamento e da fragmentação do código minimiza-se a quantidade de esforço necessária para trazer o projeto para uma nova plataforma, seja qual for o motivo para tal migração – custo, final de vida de dispositivo, entre outros.

4.5.1.4. Atualização

Com a separação do código a atualização de trechos específicos há de ser transparente para o restante do código, facilitando a manutenção e aumentando a vida útil do firmware.

4.5.2. Convenção de código

A fim de se facilitar o entendimento, o propósito e a localização de um elemento referenciado no código adotou-se uma série de convenções de código, explicadas adiante. Estas convenções foram adaptadas de convenções como a notação húngara e de convenções adotadas em grandes projetos de código aberto, como o FreeRTOS,

Os tipos básicos possíveis para variáveis foram derivados do arquivo padrão *stdint.h*. Quanto aos nomes das variáveis: o tipo de uma variável é explicitado em seu nome a partir de um prefixo. O nome propriamente dito deve seguir a convenção CamelCase, aonde as palavras que compõem o nome da variável devem ser unidas sem espaços e começar em letra maiúsculas. O Quadro 4 resume os tipos utilizados na implementação do software embarcado.

Quadro 4: Convenções para variáveis de tipos básicos

| Tipo | Tamanho (bytes) | Prefixo | Exemplo de variável |
|----------|-----------------|---------|---------------------|
| uint8_t | 1 | u8 | u8Index |
| uint16_t | 2 | u16 | u16TimeToWait |
| uint32_t | 4 | u32 | u32Deviceld |
| int8_t | 1 | i8 | i8Sum |
| int16_t | 2 | i16 | i16Difference |
| int32_t | 4 | i32 | i32Result |
| char | 1 | c | cLetter |

Fonte: autoria própria

Quanto a tipos compostos: tipos definidos no código (através de *typedefs*) foram prefixados pela letra ‘t’, seguida da letra prefixo do tipo base ao novo tipo – ‘e’ para enumerações e ‘s’ para estruturas.

The screenshot displays two examples of composite type nomenclature in a code reference tool. On the left, a struct reference for `tsActuatorTaskRequest` is shown, including its header file and data fields. On the right, an enum reference for `teActuatorTaskRequestType` is shown, including its header file and a table of its enumerators.

| Data Fields | |
|--|---|
| <code>teActuatorTaskRequestType</code> | eType The type of the request. More... |
| <code>teActuatorid</code> | eld The id of the target actuator. More... |
| <code>teActuatorTaskRequestStatus</code> | eStatus The status of the request. More... |
| <code>uint8_t</code> | u8Duration The duration of the request. More... |
| <code>uint32_t</code> | u32Id Target identity. More... |

| Enumerator | |
|--|--|
| <code>REQUEST_DEACTIVATE</code> | Request to deactivate. |
| <code>REQUEST_ACTIVATE</code> | Request to activate. |
| <code>REQUEST_FLASH_FAST_TIMED</code> | Request to toggle fast during an amount of time. |
| <code>REQUEST_FLASH_SLOW_TIMED</code> | Request to toggle slow during an amount of time. |
| <code>REQUEST_FLASH_FAST_STEADY</code> | Request to toggle fast. |
| <code>REQUEST_FLASH_SLOW_STEADY</code> | Request to toggle slow. |

Figura 18: Exemplos de nomenclatura de tipos compostos
Fonte: autoria própria

Quando o caso de ponteiros, colocou-se o prefixo ‘p’ à esquerda. Quando se tratar de vetores de dados pré-allocados (*arrays* estáticos) utilizou-se o símbolo ‘_’ como sufixo.

Para se saber a qual módulo ou grupo de implementação do software uma variável ou tipo pertence colocou-se o nome de tal grupo como primeiras palavras do nome da variável. Por exemplo, a Figura 18 mostra dois tipos definidos no código – *tsActuatorTaskRequest* e *teActuatorTaskRequestType* – que fazem parte do módulo *ActuatorTask*.

Não foram utilizadas variáveis globais compartilhadas por diferentes módulos, a não ser constantes. Foram utilizadas variáveis globais contidas ao módulo, neste caso seu nome é precedido por *prv_*, remetendo a *private*, privado ao módulo. Caso seja interessante tornar externo o acesso à variável isso é feito através de funções (*getters* e *setters* em linguagem orientada a objetos).

4.5.3. Arquitetura definida

O firmware foi dividido em quatro camadas de projeto. São elas: aplicação, funcional, adaptadora e plataforma. O infográfico da Figura 19 ilustra e descreve brevemente as camadas. O nível de abstração decresce de cima para baixo, sendo a camada de plataforma a de mais baixo nível, lidando diretamente com os registradores

do microcontrolador utilizado, enquanto a camada de aplicação implementa tarefas e ações utilizando-se das camadas inferiores a fim de atingir os objetivos desejados.

4.5.3.1. Camada plataforma

Esta camada consiste no agrupamento das bibliotecas e pacotes de código que manipulam diretamente os registradores do microcontrolador utilizado. Contém *device drivers* do dispositivo, capazes de operar seus periféricos. Por exemplo, é nesta camada que estão implementadas funções capazes de configurar as portas de GPIO, solicitar uma leitura do conversor analógico-digital ou ligar uma interrupção.

Contém também a implementação condicionada ao microcontrolador de protocolos já estabelecidos comercialmente que serão utilizados por camadas superiores, como por exemplo a implementação das rotinas necessárias para uma comunicação TCP/IP, rotinas para criptografia de dados ou para a manipulação de um sistema de arquivos.

Por fim, também deve conter a implementação de um RTOS específica para a plataforma em questão. Deve implementar a designação de funções a serem tratadas como tarefa, um escalonador para chavear os contextos de cada tarefa e mecanismos que compartilhamo de recursos padrões (semáforos, mutexes e flags).

O código que constitui esta camada é fornecido por terceiros ao projeto, seja o fabricante do dispositivo em questão ou pessoa física ou jurídica de credibilidade verificada. Deve estar acompanhado de documentação sucinta especificando as entradas, saídas e efeitos de cada trecho de código fornecido.

Os códigos contidos nesta camada são específicos ao microcontrolador ao qual fazem referência, não podendo ser reaproveitados em uma plataforma diferente.

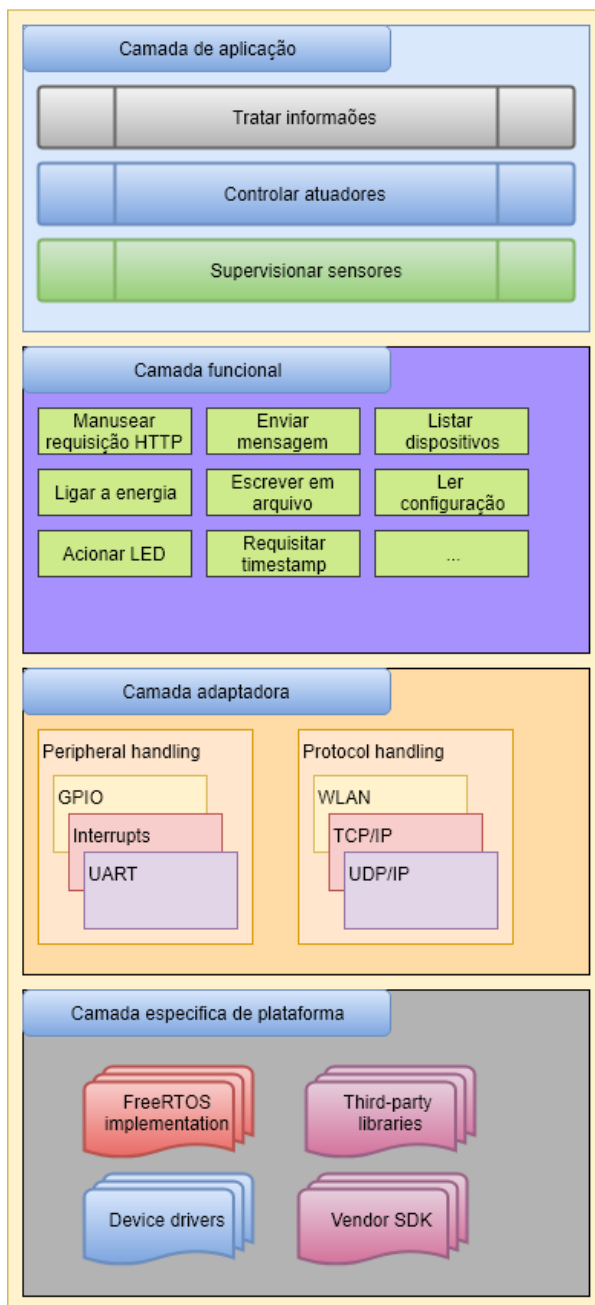


Figura 19: Ilustração da arquitetura de software
 Fonte: autoria própria

4.5.3.2. Camada adaptadora

É a camada padronizadora do projeto. Seu objetivo é tomar as definições contidas na camada inferior (de plataforma) e torná-las genéricas, abstraindo especificidades de fabricantes e de bibliotecas como por exemplo nomes de tipos e assinaturas de funções, a fim de que as camadas superiores (funcional e aplicação) não precisem saber dos detalhes de implementação física da plataforma utilizada.

Camada de aplicação

Implementa as ações do processador em face aos eventos ocorridos.

É composta pelas tarefas ativadas em resposta a estímulos fornecidos pelo usuário, por outros dispositivos ou por outras tarefas.

Camada funcional

Contém definições de procedimentos para realizar ações completas.

Não deve fazer menções ao hardware em si, mas sim aos objetos controlados pela aplicação.

Devido ao desacoplamento fornecido pela camada adaptadora deve ser possível de se executar o mesmo código fonte em plataformas distintas.

Camada adaptadora

Define tipos genéricos e funções orientadas a funcionalidades pontuais (temporização, comunicação, *input/output*...). Abstrai menções a periféricos físicos para que as camadas superiores não precisem referenciá-los.

Deve ser implementada para cada opção de microcontrolador a ser utilizada na fabricação dos produtos, sempre utilizando as mesmas assinaturas para funções e tipos definidos para manter a consistência entre as plataformas e o desacoplamento da aplicação.

Camada de plataforma

Contém definições de tipos e funções para manipular a plataforma física escolhida. É específica ao microcontrolador escolhido, fazendo menções aos seus registradores e periféricos. As bibliotecas são fornecidas pelo fabricante ou terceiros verificados.

Contém um mapeamento, onde os tipos e assinaturas de funções que as camadas superiores conhecem foram relacionados com um ou mais objetos da camada de plataforma a fim de se atingir um objetivo específico.

Por exemplo: a camada funcional necessita enviar dados pela porta serial do microcontrolador. Para isto, ela precisará configurar a porta serial e passar a ela tais dados através de funções. A camada adaptadora fornece tais funções com uma assinatura genérica, com implementação interior realizando as chamadas necessárias fornecidas pela plataforma para configurar o periférico específico e para manuseá-lo. Assim as camadas superiores precisam conhecer somente as assinaturas genéricas da camada adaptadora, abstraindo a implementação.

Como os códigos da camada adaptadora fazem referência aos contidos na camada de plataforma a camada adaptadora precisaria ser implementada tantas vezes quanto o número de microcontroladores dos quais o produto final poderá fazer uso, podendo ser tanto parcial – como por exemplo a adequação da camada a um novo microcontrolador do mesmo fabricante, de família similar porém com funcionalidades mais avançadas – quanto completa – como por exemplo a um novo microcontrolador de fabricante diferente, cuja arquitetura não possui nada em comum com a anteriormente utilizada.

4.5.3.3. Camada funcional

Esta camada agrupa o conjunto de ferramentas com as quais a aplicação realiza suas ações. Fornece definições de funções com resultados expressivos.

Sua nomenclatura evita mencionar componentes de hardware, favorecendo a evidenciação da funcionalidade implementada pela função.

Só é permitido fazer referência ao código contido na camada adaptadora.

4.5.3.4. Camada aplicação

Camada de implementação dos processos que compõem as funcionalidades do produto em questão. Utiliza as funções declaradas na camada funcional e os tipos declarados na camada adaptadora com um alto nível de abstração.

4.5.4. Módulos desenvolvidos

Aqui serão descritos os módulos desenvolvidos para desempenhar as funções necessárias a fim de se obter um sistema completo que atinja os requisitos funcionais estabelecidos na Seção 3.2. A Figura 20 ilustra a arquitetura, mostrando todos os módulos, porém com alguns vínculos omitidos a fim de se obter clareza. Os nomes dos módulos se encontram em inglês para refletir os nomes utilizados nos arquivos do código.

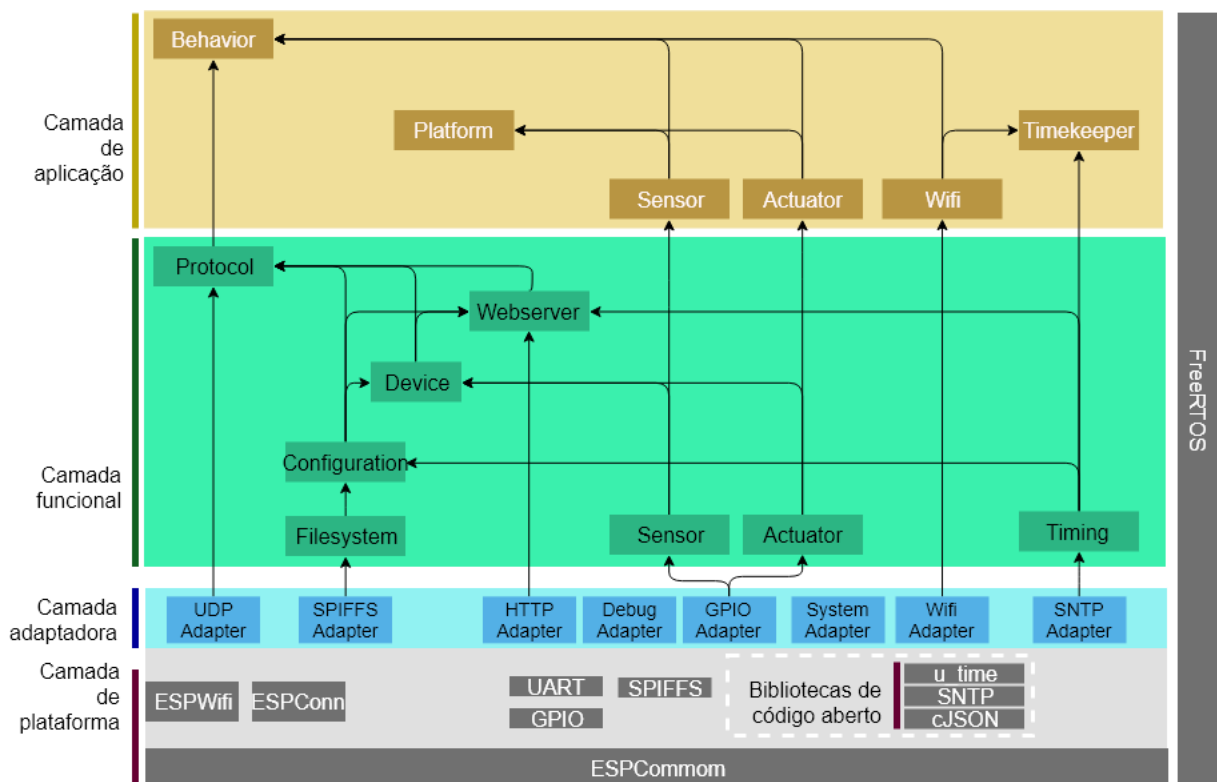


Figura 20: Vista da arquitetura com foco nos módulos desenvolvidos
Fonte: autoria própria

Não serão detalhados os tipos e chamadas de funções desenvolvidos, dando um enfoque na funcionalidade implementada pelo pedaço de software. Caso haja interesse, um link para a documentação *doxygen* do software está disponível no Apêndice A.

4.5.4.1. Módulos da camada adaptadora

Os módulos de software da camada adaptadora adaptam as funções e tipos das bibliotecas do fabricante ESP8266 e de pessoas terceiras.

4.5.4.1.1. *Tipos especiais*

Dois tipos especiais foram criados a fim de serem utilizados ao longo de todo o projeto.

O tipo *teException* é um enumerador com intuito de ser utilizado como valor de retorno de funções, enumerando as diversas falhas nas quais uma função pode incorrer. Seu uso é feito principalmente nos adaptadores UDP, wi-fi e no módulo de sistema de arquivos.

Como a fila do Free-RTOS trabalha com cópia do dado a ser enfileirado buscou-se utilizar o menor tamanho possível de mensagens a fim de não criar um lapso de tempo demasiadamente grande ao se acessar a fila. Foi então criado o tipo *tsMemQueueMessage*, composto por um identificador (do tipo *teMemQueueMessageType*) e por um ponteiro genérico. É utilizado como tipo de dado a ser inserido em todas as filas, e o enumerador de tipo nele contido serve para diferenciar as mensagens e indicar o tipo de estrutura que o ponteiro genérico aponta. Desta maneira uma fila pode receber diversos tipos de dados, tornando uma única fila o bastante para cada tarefa.

4.5.4.1.2. *Adaptador UDP (UDP Adapter)*

Este módulo foi construído para adaptar as funções relacionadas à comunicação UDP da plataforma ESP8266.

O recebimento de mensagens UDP foi implementado através de uma função que cadastra uma fila do RTOS onde as mensagens serão postas. Uma função *call-back* é cadastrada utilizando a SDK Espressif que realiza a conversão de dados em um tipo de estrutura próprio para mensagens UDP.

O envio de mensagens é realizado diretamente através de uma função que recebe o IP e a porta que se deseja enviar a mensagem.

4.5.4.1.3. *Adaptador GPIO (GPIO Adapter)*

O adaptador de GPIO contém funções a fim de configurar os modos dos pinos do ESP8266, ler o estado atual dos pinos, escrever um nível lógico nos pinos e cadastrar uma função *call-back* a ser chamada caso ocorra uma mudança de estado em um pino configurado para disparar eventos.

4.5.4.1.4. *Adaptador Wifi (Wifi Adapter)*

O adaptador de wi-fi adapta as chamadas da SDK Espressif que realizam a configuração da conexão wi-fi, seja ela como ponto de acesso ou como estação conectada a um roteador. Possui funções que verificam o estado de conexão atual e o IP adquirido pelo roteador/ponto de acesso onde está conectado.

Também define um *call-back* que é chamado sempre que o estado da conexão é alterado. Este *call-back* envia para uma tarefa - através de uma fila cadastrada - uma notificação de que a conexão *wi-fi* foi alterada.

4.5.4.1.5. *Adaptador de Sistema (System Adapter)*

O adaptador de sistema trabalha com as chamadas da SDK da Espressif relacionadas com o processador. Fornece funções que retornam o tempo de execução do programa em microssegundos, retornam quantidade de memória *heap* disponível, retornam números aleatórios e causam *reset* do microcontrolador.

4.5.4.1.6. *Adaptador de Depuração (Debug Adapter)*

O adaptador de depuração define uma macro de acesso à função *printf* a fim de depurar o programa. A macro protege a chamada com um mutex a fim de prevenir o acesso por outras tarefas enquanto uma escreve uma frase, e formata o texto passado com o nome do arquivo e o número da linha onde foi chamada.

4.5.4.1.7. *Adaptador HTTP (HTTP Adapter)*

O adaptador HTTP utiliza as funções da SDK da Espressif para abrir a porta 80 utilizando protocolo UDP, recebendo requisições e encaminhando estas para o módulo *webserver* através de um *call-back* cadastrado.

É o único adaptador que define uma tarefa, tarefa esta que bloqueia aguardando por uma requisição HTTP. Quando recebida a requisição, a tarefa a encaminha para o manuseador de requisições definido no *webserver* a fim de que este processe a requisição e monte a resposta, para que a tarefa possa enviá-la e voltar a bloquear aguardando nova requisição na porta.

4.5.4.1.8. *Adaptador SPIFFS (SPIFFS Adapter)*

O adaptador SPIFFS realiza interface com a biblioteca a fim de criar funções para criar, abrir, fechar, ler e adicionar conteúdo a arquivos. Possui uma

função que inicializa o sistema de arquivos, verificando sua disponibilidade e consistência. Caso verifique-se corrupção ou inexistência do sistema de arquivos esta função formata a área destinada ao sistema.

O adaptador também possui um mutex que deve ser reservado pela tarefa que necessite utilizar o sistema de arquivos, a fim de prevenir que uma tarefa diferente tente acessar o sistema enquanto a outra também o está acessando.

4.5.4.1.9. *Adaptador SNTP (SNTP Adapter)*

O adaptador relacionado ao protocolo SNTP define as funções de inicialização de funcionalidade - cadastrando um servidor NTP brasileiro (a.st1.ntp.br) e configurando a zona GMT de referência – e de requisição de *timestamp* – que retorna a *timestamp* caso bem-sucedido ou zero caso malsucedido.

4.5.4.2. Módulos da camada funcional

4.5.4.2.1. *Módulo sensor (Sensor module)*

O módulo sensor é responsável por inicializar os sensores da plataforma e definir um *call-back* a ser chamado quando um evento de mudança de estado é detectado. O *call-back* então passa, através de uma fila, a notificação de mudança de estado para uma tarefa competente.

4.5.4.2.2. *Módulo atuador (Actuator module)*

O módulo atuador é responsável por inicializar os atuadores da plataforma – relé e LEDs – e realizar alterações de estado nestes quando necessário.

4.5.4.2.3. *Módulo de temporização (Timing module)*

O módulo temporizador implementa funções que lidam com *timestamps*: comparação de minuto e hora em um dia, separação da *stamp* em segundos, minutos, horas, etc.

4.5.4.2.4. *Módulo de sistema de arquivos (File system module)*

O módulo sistema de arquivos é responsável por implementar chamadas que trabalham com sequências de funções do adaptador SPIFFS.

Possui duas funções, uma que lê o conteúdo de um arquivo e outra que escreve dados em um arquivo. Estas funções são composições de chamadas do

adaptador: elas tomam o mutex do sistema de arquivos, abrem o arquivo – verificando se ele existe –, realizam a ação, fecham o arquivo e devolvem o mutex.

4.5.4.2.5. *Módulo de configuração (Configuration module)*

O módulo de configuração trabalha sobre o módulo de sistema de arquivos a fim de salvar e recuperar os dados persistentes do produto – informações da rede *wi-fi*, tabelas horárias e de regras, nome do dispositivo.

4.5.4.2.6. *Módulo de dispositivo (Device module)*

O módulo de dispositivo é responsável por definir uma estrutura de dados que resume o estado atual de um dispositivo juntamente com as funções que lidam com tal estrutura, como criação, codificação e gestão de listas de dispositivos.

4.5.4.2.7. *Módulo de protocolo (Protocol module)*

O módulo de protocolo é responsável por construir mensagens de protocolo e codificá-las em e decodificá-las a partir de mensagens UDP.

4.5.4.2.8. *Módulo de servidor web (Webserver module)*

O módulo de servidor web trata as mensagens HTTP recebidas, e monta as respostas de acordo.

Uma mensagem HTTP é uma *string* terminada em zero, contendo um cabeçalho e um campo opcional de dados. A Figura 21 demonstra uma requisição HTTP simples – ela é enviada pelo navegador ao dispositivo quando se digita no campo de navegação o IP do dispositivo.

```
[code/module/webserver/webserver.c:39]New request!
GET / HTTP/1.1

Host: 192.168.0.200

Connection: keep-alive
|
Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.146 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: gsScrollPos-516=0; gsScrollPos-687=; gsScrollPos-284=0
```

Figura 21: Exemplo de requisição HTTP
Fonte: autoria própria

A primeira linha é composta pelo tipo de ação (GET neste caso), o recurso alvo (é o campo depois do endereço, como neste caso o endereço requisitado foi <ip1.ip2.ip3.ip4>/ o recurso requisitado veio vazio), e a versão do protocolo.

Os demais itens do cabeçalho ajudam o servidor a identificar o sistema que o está acessando. Neste caso, como o servidor é um microcontrolador de pequeno poder computacional – quando comparado a servidores convencionais – o restante do cabeçalho é ignorado. Por fim, o último campo em uma requisição HTTP é o campo de dados, sinalizado pela sequência ASCII “\r\n\r\n”. Como se trata de uma transação GET o campo de dados da requisição é vazio.

O servidor implementado analisa a primeira linha da requisição, detectando seu tipo e seu recurso alvo. Caso necessário, o microcontrolador processa os dados passados (no caso de transação POST). Então monta a resposta e através do adaptador HTTP a envia ao cliente.

```
POST /cgi/issue_action HTTP/1.1
Host: 192.168.0.200
Connection: keep-alive
Content-Length: 46
Origin: http://192.168.0.200
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.146 Safari/537.36
Content-Type: application/json
Accept: */*
Referer: http://192.168.0.200/
Accept-Encoding: gzip, deflate
Accept-Language: pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: gsScrollPos-516=0; gsScrollPos-687=; gsScrollPos-284=0

{"type":1,"status":true,"target":"1673449308"}
```

Figura 22: Exemplo de requisição POST

Fonte: autoria própria

A Figura 22 ilustra um exemplo de transação POST que ocorre quando o usuário deseja ligar um aparelho conectado a um dispositivo. Observe os dados enviados pela interface para o servidor na última linha da requisição. Para trocar dados entre servidor e cliente foi escolhida a notação JSON (*Javascript Object Notation*), uma notação que busca fácil entendimento da informação quando lida por humanos e fácil geração e interpretação por máquinas. No exemplo, a interface web deseja executar o manipulador *issue_action* (“disparar ação”, verificável na primeira linha do cabeçalho) com os dados: tipo igual a um (requisição de atuador), novo status igual a *true* (equivalente a ligar) e o alvo é o dispositivo cujo identificador vale 1673449308.

No código embarcado uma biblioteca de código aberto chamada *cJSON* foi utilizada. Esta notação é manipulada nativamente pelo código *Javascript* da página web do cliente.

4.5.4.3. Módulos da camada de aplicação

Os módulos da camada de aplicação são as tarefas propriamente ditas. Todas as tarefas do sistema são consideradas da camada de aplicação, salvo a tarefa HTTP que é implementada no adaptador HTTP devido à sua proximidade com a interface TCP. A Figura 23 é um diagrama simplificado das relações entre as tarefas e os recursos a serem controlados. Recursos e módulos que não aparecem no diagrama, como o sistema de arquivos e a configuração do sistema, são acessados pela maioria das tarefas e foram omitidos a fim de manter clareza.

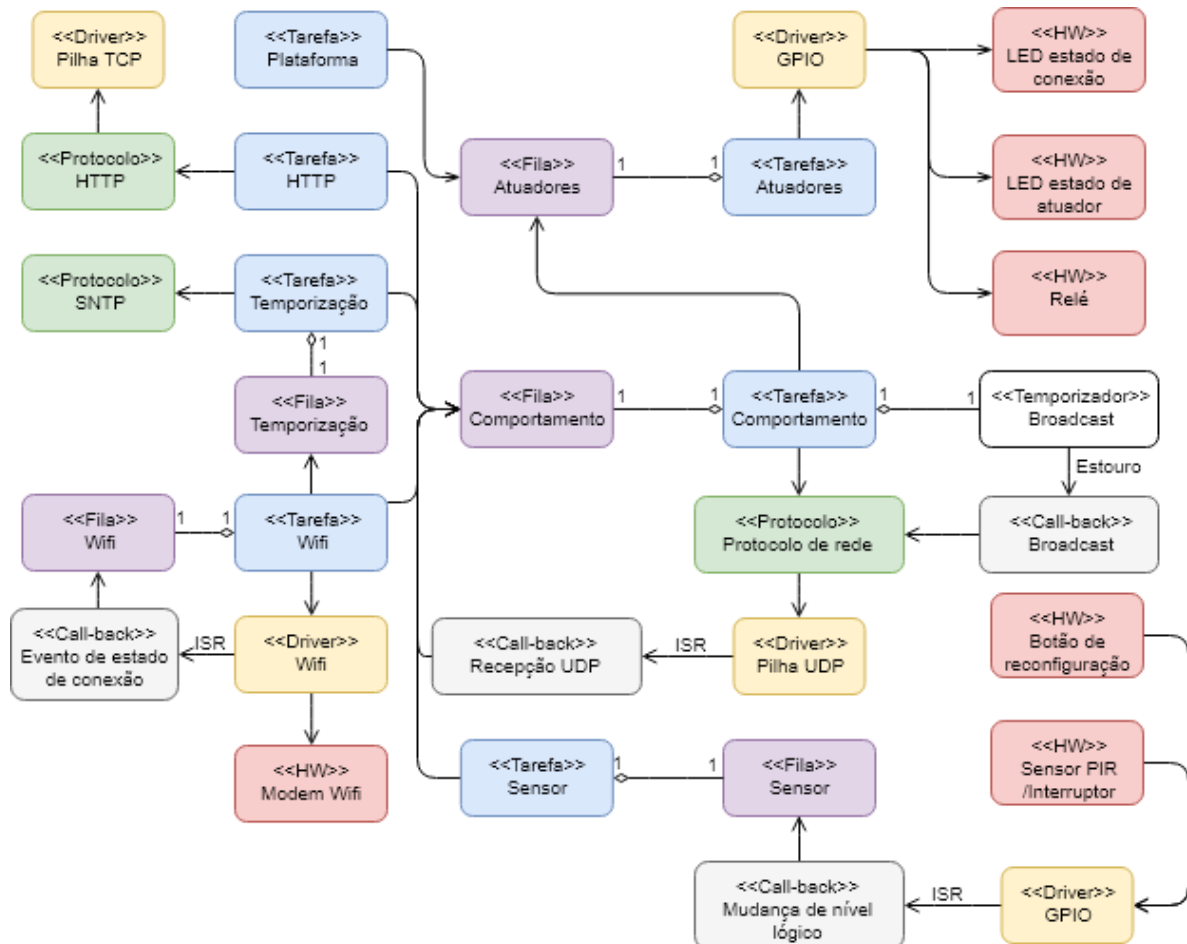


Figura 23: Diagrama de componentes do sistema
Fonte: autoria própria

4.5.4.3.1. *Tarefa wi-fi*

A tarefa wi-fi é responsável pela conexão wi-fi do sistema. Ao inicializar ela verifica a configuração do dispositivo na memória flash e decide se deve configurar o controlador como ponto de acesso (caso o dispositivo esteja em modo de configuração) ou se deve se conectar à WLAN do usuário, caso já tenha sido configurada previamente.

Caso seja a primeira vez que a tarefa tente se conectar à WLAN do usuário (usuário acabou de passar os dados da WLAN para o dispositivo) e a conexão falhe, a tarefa causa um reset no dispositivo fazendo-o retornar ao modo de configuração.

Caso não seja a primeira vez que a tarefa tente se conectar à WLAN do usuário (retorno de queda de energia, por exemplo) a tarefa continua tentando incessantemente se conectar à WLAN.

A tarefa então repousa bloqueada em sua fila, aguardando mudanças de estado na conexão wi-fi. Quando uma mudança é percebida, a tarefa avisa através de mensagens de fila as tarefas que têm interesse no estado da wi-fi: temporizadora e comportamental.

4.5.4.3.2. *Tarefa de sensor*

A tarefa de sensor é responsável pelo repasse das mensagens do *call-back* da mudança de estado de GPIO, disparado por eventos em sensores, para a tarefa comportamental.

4.5.4.3.3. *Tarefa de plataforma*

A tarefa de plataforma é responsável por supervisionar o estado do atuador e da conexão wi-fi por *polling* periodicamente, enviando mensagens de controle dos atuadores LED para a tarefa de atuadores a fim de representar visualmente no dispositivo o estado do sistema.

4.5.4.3.4. *Tarefa de atuador*

A tarefa de atuador é responsável por mudar o estado dos atuadores – relé e LEDs – de acordo com mensagens recebidas por outras tarefas. Permite o estabelecimento de estados estacionários – ligado e desligado – bem como estados osciladores – utilizado para piscar os LEDs.

4.5.4.3.5. Tarefa de temporização

A tarefa de temporização é responsável por comparar uma *timestamp* atual com a tabela horária gravada na memória, e caso a comparação com algum valor seja verdadeira altera o estado do relé para aquele gravado na linha da tabela horária.

A tabela horária é uma estrutura composta por quatro entradas, e cada entrada é composta por uma hora e minuto do dia (representada como somatório de minutos desde a hora 00:00) e uma ação (ligar ou desligar) que deve ocorrer neste horário. Esta tabela é independente em cada dispositivo na rede, podendo o usuário reprogramar a tabela de um ou mais dispositivos isoladamente através da interface.

A tarefa aguarda, ao iniciar, uma mensagem da tarefa wi-fi indicando que a conexão com a rede do cliente foi bem-sucedida. Então a tarefa tenta conseguir uma *timestamp* através do adaptador SNTP. Quando consegue, utiliza as funcionalidades do módulo de temporização para trabalhar com a *timestamp* e compará-la com os valores da tabela horária, emitindo mensagens para a tarefa atuadora quando ocorrer uma correspondência com algum valor válido da tabela.

4.5.4.3.6. Tarefa comportamental

A tarefa comportamental é a mais importante do sistema. É responsável pelo comportamento em rede, processando os dados gerados por outras tarefas, recebidos de outros dispositivos e recebidos pela interface do usuário.

Esta tarefa tem trechos de execução separados pelo modo que o dispositivo está operando – mestre ou escravo. Para decidir o modo em que está, o dispositivo primeiro verifica através do adaptador wi-fi qual o estado da conexão. Se não estiver conectado, o modo offline é estabelecido e somente os eventos de sensores e mensagens da tarefa wi-fi – que indicariam um reestabelecimento da conexão – são recebidos. Caso conectado, a tarefa bloqueia em sua fila aguardando uma mensagem UDP com um *timeout* aleatório. Caso o *timeout* ocorra, significa que não há mestre na rede e que o dispositivo deve assumir este papel. Caso receba a mensagem o dispositivo assume o papel do escravo e reporta ao mestre. Esta rotina é ilustrada pelo diagrama de atividades da Figura 24.

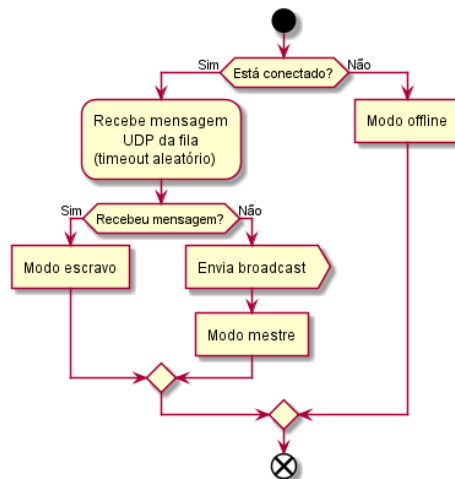


Figura 24: Diagrama de atividades da rotina de detecção de modo
Fonte: autoria própria

Quando decidido o modo a execução da tarefa segue por dois trechos: tratamento de mensagens da fila e tratamento de mensagens da rede. A Figura 25 ilustra as atividades da tarefa comportamental em um dispositivo conectado à rede.

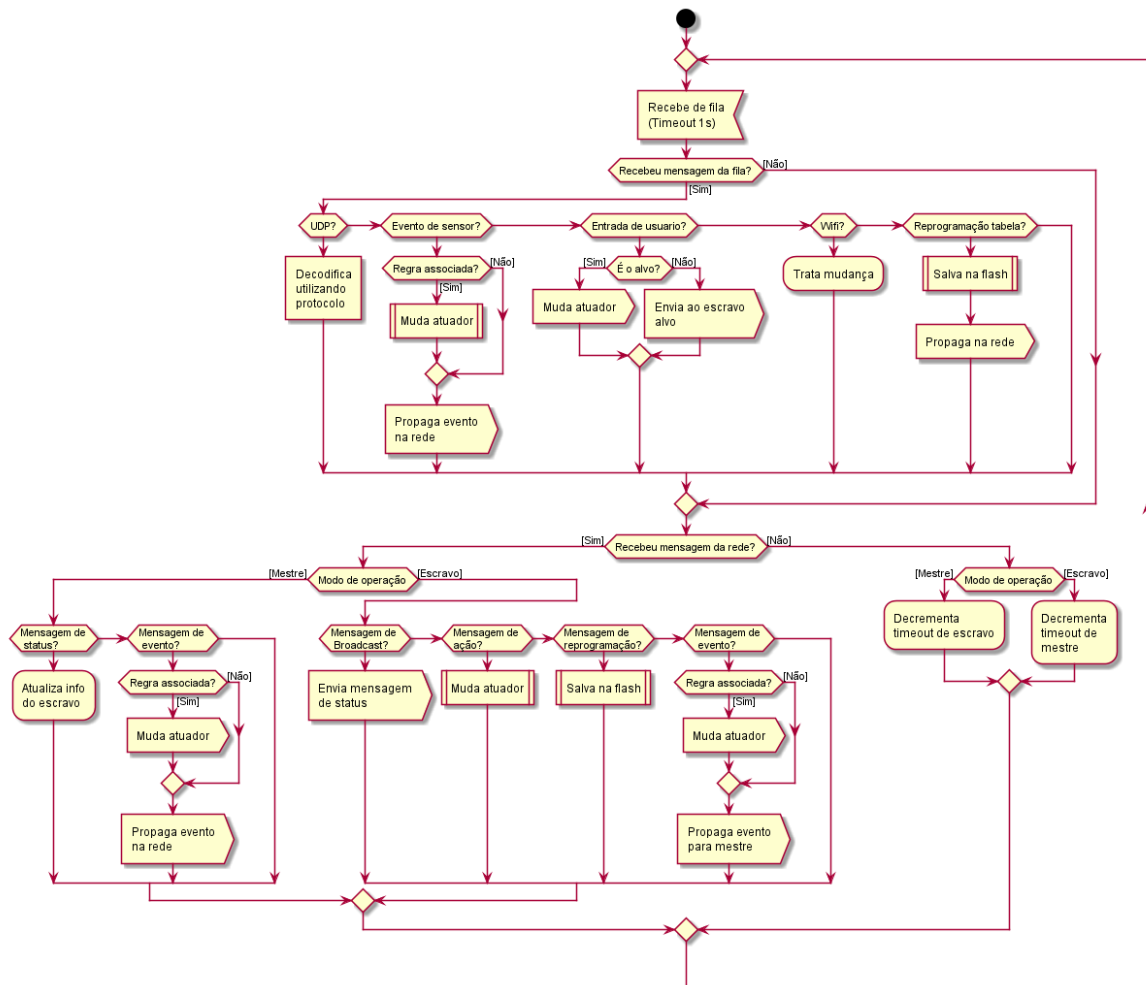


Figura 25: Atividades da tarefa comportamental
Fonte: autoria própria

A tarefa comportamental bloqueia por um segundo esperando uma nova mensagem em sua fila. Quando a mensagem chega, seu conteúdo é interpretado conforme seu identificador. As mensagens possíveis são as seguintes:

- Mensagem UDP: é decodificada utilizando o módulo protocolo e guardada para tratamento no trecho seguinte;
- Mensagem de evento local: é verificado se o evento está associado a alguma regra programada que afete o dispositivo, em seguida o evento é propagado na rede (do mestre aos escravos, caso o dispositivo seja mestre; ou do escravo ao mestre caso o dispositivo seja escravo);
- Mensagem de entrada de usuário: mensagem originada pela tarefa HTTP contendo uma ação emitida pelo usuário. Se o dispositivo for o alvo deve de executar a ação, caso contrário encaminha ao escravo pertinente;
- Mensagem wi-fi: simboliza mudança no estado da rede. Caso tenha ocorrido uma desconexão a tarefa passa para o modo *offline*. Posteriormente caso tenha ocorrido uma conexão bem-sucedida a tarefa executa a função de decisão de modo;
- Reprogramação de tabela (horária ou de regras): mensagem originada pela tarefa HTTP contendo uma nova tabela a ser reprogramada. A tabela de regras é comum a todos os dispositivos e deve ser propagada do mestre aos escravos, enquanto a tabela horária é única para cada dispositivo e deve ser propagada somente ao dispositivo alvo, quando aplicável.

A tabela de regras é o elemento que permite que um dispositivo responda com uma mudança do seu atuador a um evento de um dispositivo qualquer da rede. É programada pelo usuário através da interface web e é igual entre todos os dispositivos da rede. Ao receber um evento de sensor, a tarefa comportamental irá utilizar o identificador do dispositivo que originou o evento, juntamente com o nível do evento e com seu número identificador para verificar se há alguma regra cadastrada com tais dados. Caso positivo, a tarefa comportamental envia à tarefa de atuador uma mensagem requisitando mudança para o estado descrito pela regra correspondente – carga energizada ou desligada.

Caso a tarefa comportamental tenha recebido uma mensagem UDP, seu conteúdo é decodificado e executado de maneira semelhante às mensagens de fila.

Caso seja um escravo e tenha recebido um *broadcast* do mestre, o dispositivo responde com uma mensagem de status e reseta um temporizador de *timeout* de mestre, que caso estoure fará com que o dispositivo escravo execute a rotina de decisão de modo.

Caso seja um mestre e tenha recebido uma mensagem de status, a tarefa cadastra o novo dispositivo escravo ou atualiza a entrada do dispositivo em sua memória, resetando um temporizador individual que caso estoure por falta de mensagens de status leva a conclusão de que o dispositivo deixou a rede.

4.5.5. Discussão da configuração das tarefas

A SDK da Espressif permite que uma tarefa seja definida com prioridade com um valor igual a um número entre um (menor prioridade) e dez (maior prioridade).

A tarefa plataforma foi inicializada com prioridade igual a 2, pois é responsável somente por atualizar os LEDs da placa, que não tem nenhum requisito de tempo rigoroso a ser cumprido.

As tarefas de sensor, de atuador e de temporização foram configuradas com prioridade igual a 3, pois também não possuem requisitos temporais rigorosos. Desde que sejam permitidas a rodar alcançarão seus objetivos. Com relação a temporização e a comparação da tabela horária, desde que a tarefa de temporização seja permitida executar uma vez a cada minuto poderá realizar a inspeção da tabela e comparação com a hora e minuto atual a fim de verificar se deve tomar alguma ação programada.

A tarefa de comportamento possui valor de prioridade igual a 6, pois apesar de ser importante que ela trate os dados da rede e as requisições do usuário a importância e a baixa latência desejada para as duas outras tarefas seguintes a coloca como terceira prioridade mais alta.

A tarefa servidora HTTP ganha prioridade 7 a fim de reduzir a latência com que as requisições HTTP sejam respondidas.

A tarefa controladora de wi-fi recebe prioridade 8 a fim de que mudanças no estado da conexão wi-fi sejam percebidas o quanto antes e propagadas às demais tarefas.

4.6. Projeto web

O projeto web compreendeu o design da interface a ser acessada pelo usuário via navegador. Para simplificar o projeto decidiu-se por projetar duas páginas somente – a página de configuração e a página de controle do sistema – e realizar a dinâmica das páginas através de *scripts* em linguagem Javascript.

O estilo das páginas (cores, fontes, disposições) foi realizado através de descrições em CSS, o que possibilitou um design responsivo – diz-se que uma página possui design responsivo quando a disposição de seus elementos é atualizada de maneira a tornar sua navegação confortável de acordo com o dispositivo de onde está sendo acessada.

4.6.1. Considerações gerais

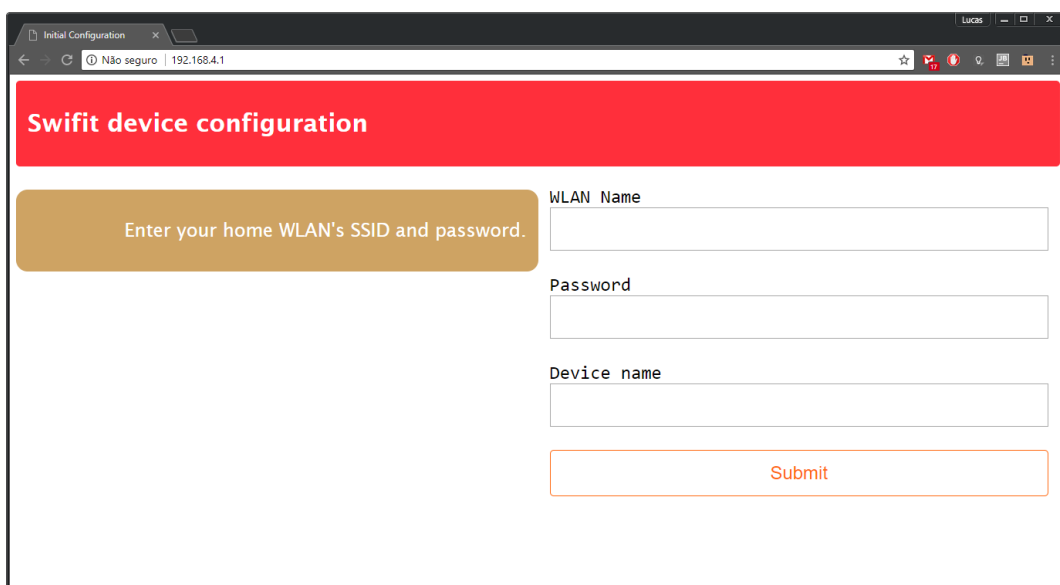
Como definido anteriormente, apenas duas páginas web são criadas e em um dado modo de operação (em configuração ou em funcionamento) a página web propriamente dita é carregada uma única vez a partir do dispositivo. Porém, em modo de funcionamento normal, o usuário deve de ser capaz de verificar mudanças e enviar dados ao dispositivo sem que a página web inteira seja recarregada. Para isso utilizou-se a tecnologia AJAX (*Asynchronous Javascript And XML*), onde o código Javascript rodando no navegador do cliente pode, através de requisições HTTP, trocar dados com o servidor sem que a página seja recarregada – a requisição AJAX faz com que o navegador crie uma requisição HTTP, porém não tente interpretar o conteúdo da resposta como uma nova página web.

```
[code/module/webserver/handlers/devices_timing.c:37]JSON Created: [{"name": "Socket1", "id": "1673449308", "times": [{"time": "1291", "enabled": true, "action": true}, {"time": "0", "enabled": false, "action": false}, {"time": "0", "enabled": false, "action": false}, {"time": "0", "enabled": false, "action": false}]}]
```

Figura 26: Tabela de temporização de um dispositivo em formato JSON
Fonte: autoria própria

Para os dados trocados nestas requisições foi escolhida a notação JSON (*Javascript Object Notation*). É uma notação compacta baseada em texto em que os dados transmitidos podem ser facilmente interpretados tanto por humanos como máquinas. A Figura 26 é a cópia de uma mensagem de depuração que mostra o conteúdo de uma mensagem JSON representando a tabela de temporização com uma entrada ativa para acionar o relé diariamente no minuto 1291 de um dispositivo de nome “Socket1” com número identificador 1673449308.

4.6.2. Página de configuração



The screenshot shows a web browser window with the address bar displaying '192.168.4.1'. The page title is 'Initial Configuration'. The main content area has a red header with the text 'Swift device configuration'. Below this is a brown box with the text 'Enter your home WLAN's SSID and password.'. To the right of this box are three input fields: 'WLAN Name', 'Password', and 'Device name'. At the bottom right is a 'Submit' button.

Figura 27: Página de configuração
Fonte: autoria própria

A página de configuração – ilustrada na Figura 27 – tem o intuito de capturar a entrada dos dados da WLAN do usuário, bem como o nome personalizado que o usuário deseja dar ao produto. Quando o usuário pressiona o botão *submit*, um evento é gerado e sua função é de enviar através de uma requisição AJAX um objeto JSON contendo os dados entrados. Aparecerá uma mensagem na tela informando o usuário que deve checar o dispositivo – o modo de piscar dos LEDs deve indicar o sucesso ou fracasso ao tentar conectar à rede.

Caso fracasse, ao retornar a esta página, o usuário poderá verificar o motivo de fracasso: rede não encontrada, senha incorreta ou motivo não identificado.

4.6.3. Página de controle

A página de controle é dividida em três subpáginas – todas contidas em um mesmo documento HTML, porém somente uma é mostrada a cada momento de acordo com o elemento selecionado na barra de navegação na parte superior.

4.6.3.1. Subpágina de dispositivos

A subpágina de dispositivos é responsável por mostrar o estado atual de todos os dispositivos na rede.

A cada um segundo ela se comunica com o dispositivo mestre requisitando via AJAX um objeto JSON contendo a lista de dispositivos da rede, e com as informações recebidas cria elementos HTML que exibem o estado dos dispositivos. Através de uma chave neste elemento, o usuário pode enviar a um dispositivo na rede um comando de ligar ou desligar a carga a ele conectada.

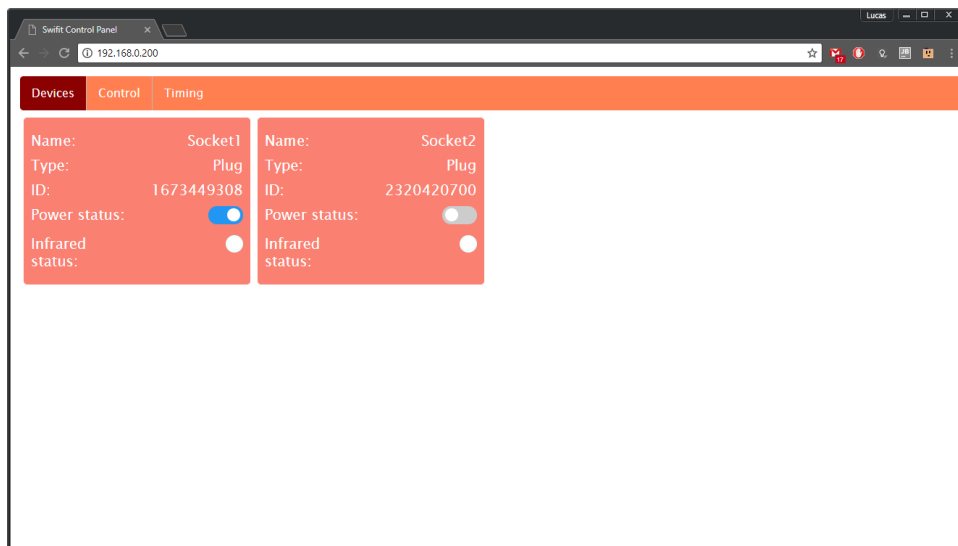


Figura 28: Subpágina de dispositivos
Fonte: autoria própria

A Figura 28 demonstra a subpágina de dispositivos de uma rede com duas tomadas inteligentes: a primeira, de nome “Socket1”, com sua carga acionada e o segundo, “Socket2”, com sua carga desativada.

4.6.3.2. Subpágina de controle

A subpágina de controle é utilizada para criar as regras da rede. Possui quatro botões: um que carrega as regras instaladas na rede, um que adiciona uma nova regra à página, um que limpa todas as regras da página e um que envia as regras atualmente na página para os dispositivos na rede, reprogramando-os.

Uma regra de rede relaciona um dispositivo disparador, um nível lógico de sensor disparador, dispositivo alvo e uma ação alvo. Quando detectada uma mudança de nível lógico para o nível disparador no sensor do dispositivo disparador, a ação alvo deve ser executada pelo dispositivo alvo. A Figura 29 mostra a subpágina de controle para a mesma rede ilustrada pela Figura 28.



Figura 29: Página de controle
Fonte: autoria própria

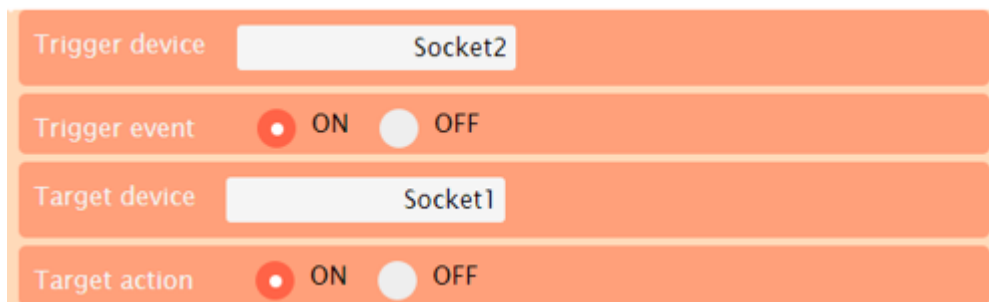


Figura 30: Exemplo de regra
Fonte: autoria própria

Por exemplo, a regra em evidência na Figura 30 irá ligar o aparelho conectado ao “Socket1” assim que o sensor PIR do “Socket2” detectar presença.

4.6.3.3. Subpágina de temporização

A subpágina de temporização é responsável por mostrar a tabela horária de cada um dos dispositivos presentes na rede. Possui um botão para carregar as tabelas atuais da rede, e permite ao usuário editar e salvar as tabelas dos dispositivos.

Uma tabela de um dispositivo é composta de quatro entradas (linhas). Cada linha tem uma chave que diz se a regra é válida ou não, o horário da regra e a ação que deve ser executada no aparelho conectado – ligar ou desligar.

A Figura 31 ilustra a subpágina de temporização para uma rede de dois dispositivos. Ambos os dispositivos possuem somente uma regra de tempo ativa (a primeira), ambos com o mesmo horário de disparo (21h 31min). O dispositivo “Socket1” irá desligar sua carga quando o horário de disparo chegar, enquanto “Socket2” irá ligar a carga.

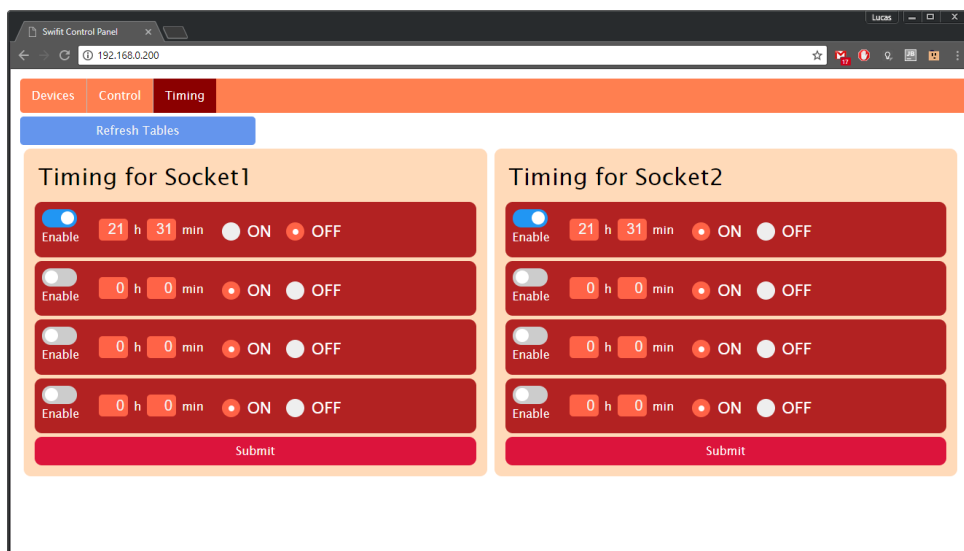


Figura 31: Subpágina de temporização
Fonte: autoria própria

5. DISCUSSÃO DOS RESULTADOS

O objetivo geral do projeto - entregar um conjunto de protótipos eletrônicos, alinhados ao conceito de IoT, que controlem a alimentação de utensílios eletroeletrônicos de maneira automática seguindo uma programação de um usuário – foi alcançado com sucesso.

Os artefatos desejados foram descritos no Capítulo 3, seguindo a metodologia aprendida nas aulas de Engenharia de Produto, e a partir de tal descrição foram estipulados, na Seção 3.2, os requisitos que guiaram as funcionalidades dos produtos – de acordo com o que foi visto nas aulas de Sistemas Embarcados.

Um estudo das opções de tecnologias disponíveis foi realizado em seguida, descrito na Seção 4.1, e baseando-se em conceitos aprendidos nas aulas de Eletrônica Básica, Microcontroladores, Sistemas Operacionais, Sistemas Embarcados e Redes de Computadores foram eleitas as tecnologias mais interessantes ao projeto – eleição guiada pelos requisitos estipulados anteriormente.

O projeto e sua implementação foram realizados em seguida, processo descrito nas Seções 4.3 (hardware), 4.4 (rede), 4.5 (software) e 4.6 (web). Um hardware foi projetado e montado segundo os conhecimentos adquiridos nas disciplinas Eletrônica Básica, Semicondutores de Potência e Fabricação Eletrônica. O software embarcado foi implementado utilizando-se conhecimentos de Fundamentos de Programação, Estruturas de Dados e Sistemas Embarcados. Com o desenvolvimento modular e códigos de depuração foi possível de se entregar um software estável, sem vazamentos de memória ou travamentos.

Ao integrar as partes desenvolvidas – hardware e software – obteve-se um protótipo que cumpre com os requisitos estipulados na Seção 3.2, tornando o projeto bem-sucedido. A Figura 32 é uma composição de três fotos dos protótipos desenvolvidos ao longo deste projeto. A foto da esquerda representa o protótipo do produto tomada inteligente, a foto do centro representa o protótipo do produto interruptor e a foto da direita mostra em detalhe a montagem do transformador e da placa de circuito impresso dentro da caixa do protótipo do interruptor.

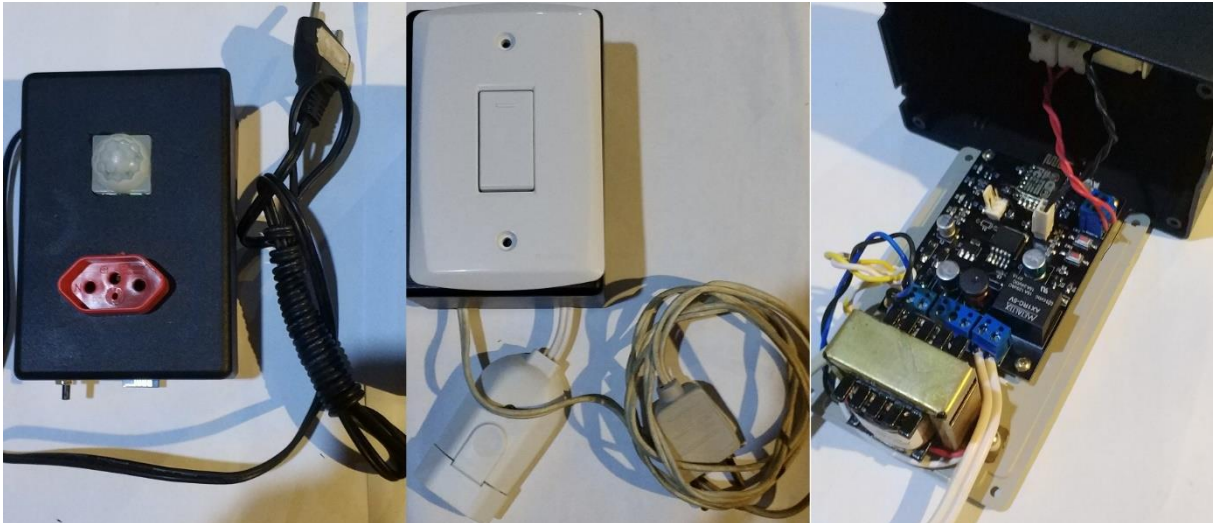


Figura 32: Fotografias dos protótipos desenvolvidos
Fonte: autoria própria

5.1. Pontos fortes

O projeto implementou com sucesso uma rede de dispositivos conectados via WLAN, com topologia estrela onde existe um dispositivo mestre e vários escravos. Proporcionou uma experiência de projeto de produto completo, com levantamento de requisitos, projeto e montagem de hardware e desenvolvimento de software. Proporcionou um contato com o desenvolvimento de páginas web e a implementação embarcada de um servidor HTTP em um microcontrolador de porte relativamente pequeno. Também proporcionou noções dos desafios de se projetar uma rede de dispositivos – delegação de atividades, dispositivos mestres, descobrimento de dispositivos e repasse de informações.

5.2. Comparação com produtos já existentes

Para fins de comparação foram escolhidos o interruptor inteligente (BELKIN, 2018) e o *plug* de tomada inteligente (BELKIN, 2018) da linha WeMo da empresa Belkin e o *plug* de tomada inteligente (ITEAD, 2018) da linha Sonoff da ITEAD. Ambos funcionam utilizando tecnologia *wi-fi*, sendo controlados através de aplicativos para os sistemas operacionais IOS e Android – possuindo seu acesso limitado a um *smartphone* ou *tablet*. Podem ser acessados fora da WLAN onde os dispositivos estão conectados. Todos esses produtos listados fornecem funcionalidade de programação de horários para ligar e desligar a carga, mas nenhum possui a funcionalidade de se utilizar os eventos de um elemento sensor de um dispositivo para disparar ações ao longo da rede. Os dispositivos Sonoff possuem a

funcionalidade de cena dinâmica – onde podem ser agrupados dispositivos a fim de que com um único comando várias ações sejam executadas – e de contagem regressiva – onde a carga é energizada durante um tempo determinado.

A título de curiosidade foi adquirido um produto da Sonoff - um elemento chaveador de custo reduzido de nome Sonoff Basic (ITEAD, 2018). Ele possui funcionalidades idênticas àquelas do *plug* inteligente Sonoff, diferindo somente no encapsulamento: ao invés de um conector de tomada têm-se bornes para conexão de fios decapados.

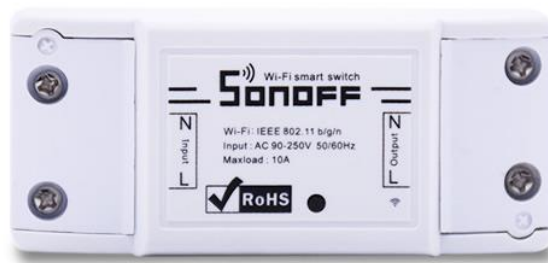


Figura 33: Sonoff Basic
Fonte: Sonoff Basic (ITEAD, 2018)

A Figura 33 é uma fotografia do produto Sonoff Basic, cuja construção é compreendida por uma caixa de plástico ABS, um botão de propósito múltiplo, um LED indicador de estado de conexão (invisível na parte exterior, porém uma área circular na caixa de plástico com grossura reduzida deixa a luz passar parcialmente quando o LED está acesso) e parafusos para fixação dos cabos dentro dos bornes.

Verificou-se que o microcontrolador utilizado pelo produto é o mesmo utilizado neste protótipo (ESP8266EX). Também é notável um vão separando as linhas de alta tensão, elemento utilizado para se obter uma maior isolamento e segurança que certamente poderia ser incluído em uma revisão futura do protótipo aqui implementado.

5.3. Problemas encontrados

A falta de um depurador que possibilitasse a inspeção da memória e a execução do programa em passos dificultou o projeto, porém não o tornou impossível.

Através do uso da porta serial como meio de depuração foi possível de se resolver problemas como a chegada de dados corrompidos, estouros em pilhas e acessos a áreas reservadas na memória.

5.4. Melhorias vislumbradas

Para a versão de produção da placa algumas melhorias são aplicáveis:

- Investir em um projeto de fonte chaveada com tensão de entrada maior, a fim de não se necessitar um transformador abaixador da tensão da rede. Como o projeto da fonte e a fabricação ou compra dos transformadores envolvidos pode custar caro, buscar projetos de fontes já prontas com empresas terceirizadas é uma alternativa a ser investigada;
- Reduzir o espaçamento e o tamanho dos componentes utilizados. Nesta montagem utilizou-se componentes SMD de tamanho 0805 com espaçamento para solda manual, enquanto para uma fabricação comercial poderia ser utilizado o tamanho 0603 ou 0402;
- Substituir os botões de reset e gravação da placa, juntamente com o conector para depuração, por *pads* (ilhas de cobre sem máscara de solda) de contato para cama de pregos a fim de dificultar o acesso aos modos de gravação e depuração e a fim de agilizar o processo de manufatura através do uso de equipamentos dedicados.

Para o software embarcado:

- Explorar outros SDKs disponíveis na comunidade, com versões mais atualizadas de RTOS e com melhor documentação;
- Implementar a funcionalidade de diário de eventos, onde o microcontrolador pode escrever em sua memória informações como data e hora de mudança de conexão com a WLAN, data e hora de programação de tabela, data e hora de acionamento de carga, entre outras possibilidades;
- Criar um protocolo de fabricação através de interface serial ou porta TCP/UDP dedicada, a fim de programar em fábrica itens como identificador de produto, data de fabricação e também possibilitar a inspeção do estado atual e do diário acima mencionado do produto,

possibilitando testes e a depuração de problemas que venham a ocorrer em campo;

- Implementar a funcionalidade de upgrade de software embarcado, possibilitando uma melhoria contínua após o produto ter sido lançado no mercado;

Para a interface:

- Criar um servidor a fim de servir a interface para o cliente, possibilitando o acesso tanto na WLAN onde os produtos estão instalados quando fora dela, e retirando do microcontrolador a carga de processamento relacionada;
- Criar aplicativos para cada plataforma a fim de se ter uma solução nativa para cada plataforma onde o usuário possa desejar acessar a interface.

REFERÊNCIAS

ADAFRUIT INDUSTRIES. PIR Motion Sensor. Disponível em: <<https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf>>. Acesso em: 19 abr. 2018.

AI-THINKER. ESP-01/07/12 Series Modules User's Manual V1.1. Disponível em: <http://wiki.ai-thinker.com/_media/esp8266/esp8266_series_modules_user_manual_v1.1.pdf>. Acesso em: 16 abr. 2018.

ALMEIDA, Alexandre Vaz de. Implementação de um sistema de automação residencial modular sem fio: módulo periférico. Disponível em: <http://www.tcc.sc.usp.br/tce/disponiveis/18/180500/tce-23042010-155834/publico/Almeida_Alexandre_Vaz_de_2.pdf>. Acesso em: 1 jun. 2018.

BEGHINI, Lucas Bragazza. Automação residencial de baixo custo por meio de dispositivos móveis com sistema operacional Android. Disponível em: <<http://www.tcc.sc.usp.br/tce/disponiveis/18/180450/tce-04022014-152853/?&lang=br>>. Acesso em: 1 jun. 2018

BELKIN. WeMo® Switch Smart Plug. Disponível em: <<http://www.belkin.com/us/p/P-F7C027/>>. Acesso em: 19 abr. 2018.

BELKIN. WeMo® Wi-Fi Smart Light Switch. Disponível em: <<http://www.belkin.com/us/p/P-F7C030/>>. Acesso em: 19 abr. 2018.

BENCHOFF, Brian. Creating a PCB in everything: KiCAD: part 1. Disponível em: <<https://hackaday.com/2016/11/17/creating-a-pcb-in-everything-kicad-part-1/>>. Acesso em: 14 mar. 2018.

D-LINK. DIR-826L Datasheet. Disponível em: <http://us.dlink.com/wp-content/uploads/2014/03/DIR-826L_DATASHEET_1.00_EN.pdf>. Acesso em: 07 mai. 2018.

ESPRESSIF SYSTEMS. Ltda. ESP8266EX Datasheet. Disponível em: <https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>. Acesso em: 14 mar. 2018.

ESPRESSIF SYSTEMS. Ltd. ESP8266 System Description. Disponível em: <https://www.espressif.com/sites/default/files/documentation/0b-esp8266_system_description_en.pdf>. Acesso em: 14 mar. 2018.

ITEAD. S20 Smart Socket. Disponível em:

<http://sonoff.itead.cc/en/products/residential/s20-socket>>. Acesso em: 19 abr. 2018.

ITEAD. SONOFF Basic. Disponível em: <

<http://sonoff.itead.cc/en/products/sonoff/sonoff-basic>>. Acesso em: 19 abr. 2018.

TEXAS INSTRUMENTS, Ltda. LM2596 SIMPLE SWITCHER® Power

Converter 150-kHz 3-A Step-Down Voltage Regulator. Disponível em: <

<http://www.ti.com/lit/ds/symlink/lm2596.pdf>>/ Acesso em: 19 abr. 2018.

APÊNDICE A – Código fonte e referências externas

O código fonte desenvolvido para este projeto está disponível no link <https://github.com/LucasPetrelli/swifit>, junto com a documentação *doxygen* (abrir o arquivo `index.html` na pasta *doxygen/html*).