UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA

Arthur Alves

PROJETO DE UM SISTEMA DE CONTROLE REMOTO DE ILUMINAÇÃO RESIDENCIAL VIA INTERNET

Florianópolis
2022

Arthur Alves

PROJETO DE UM SISTEMA DE CONTROLE REMOTO DE ILUMINAÇÃO RESIDENCIAL VIA INTERNET

Trabalho de conclusão de curso submetido ao Departamento de Engenharia Elétrica e Eletrônica da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia Eletrônica.
Orientador: Prof. Richard Demo Souza, Dr.

Florianópolis
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Alves, Arthur
Projeto de um sistema de controle remoto de iluminação residencial via internet / Arthur Alves ; orientador, Richard Demo Souza, 2022.
36 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Eletrônica, Florianópolis, 2022.

Inclui referências.

1. Engenharia Eletrônica. 2. IoT. 3. Microcontroladores. 4. RS-485. 5. Automação. I. Souza, Richard Demo. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Eletrônica. III. Título.

Arthur Alves

Projeto de um sistema de controle remoto de iluminação residencial via internet

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Engenharia Eletrônica e aprovado em sua forma final pelo Curso de Graduação em Engenharia Eletrônica.

Documento assinado digitalmente
Fernando Rangel de Sousa
Data: 19/12/2022 10:59:28-0300
CPF: ***.649.114-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Fernando Rangel de Sousa, Ph.D
Coordenador do Programa

Banca examinadora:

Documento assinado digitalmente
Richard Demo Souza
Data: 16/12/2022 14:23:04-0300
CPF: ***.267.379-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Richard Demo Souza, Dr.
Orientador

Documento assinado digitalmente
BARTOLOMEU FERREIRA UCHOA FILHO
Data: 19/12/2022 20:37:14-0300
CPF: ***.362.114-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Bartolomeu Ferreira Uchôa Filho, Ph.D
Universidade Federal de Santa Catarina

Documento assinado digitalmente
MARIO DE NORONHA NETO
Data: 20/12/2022 09:29:45-0300
CPF: 003.859.519-22
Verifique as assinaturas em https://v.ifsc.edu.br

Prof. Mario de Noronha Neto, Dr.
Instituto Federal de Santa Catarina

Florianópolis, 16 de dezembro de 2022.

Este trabalho é dedicado à minha família.

AGRADECIMENTOS

Agradeço a Deus pelo dom da vida, à minha esposa Paola pela companhia e apoio e aos meus pais, que foram minha base até aqui. Também sou muito grato ao professor Richard por todos os ensinamentos durante a graduação e pela atenciosa orientação deste trabalho.

“Se pude enxergar mais longe, foi porque me apoiei em ombros de gigantes.”
(Isaac Newton, 1675)

RESUMO

Este trabalho apresenta o desenvolvimento completo de um sistema de controle remoto de iluminação residencial via internet. Esse sistema é controlado através de uma página web que pode ser acessada a partir de computadores, celulares ou tablets, oferecendo flexibilidade aos usuários. A página envia comandos a um back-end escrito em NodeJS, que redireciona as informações a um dispositivo central, instalado no ambiente cuja iluminação está sendo controlada. Essa central está conectada, via um barramento que segue o padrão RS-485, a diversos módulos auxiliares, que são responsáveis por controlar as lâmpadas. Neste documento serão apresentados os fundamentos, etapas de projeto e os resultados obtidos ao final do trabalho.

Palavras-chave: Controle de iluminação. IoT. Microcontroladores. RS-485. Desenvolvimento web.

ABSTRACT

This work presents the complete development of a residential lighting remote control system via internet. This system is controlled through a web page that can be accessed from computers, cell phones or tablets, offering flexibility to users. The page sends commands to a backend written in NodeJS, which redirects the information to a central device, installed in the environment whose lighting is being controlled. This central is connected, via a bus that follows the RS-485 standard, to several auxiliary modules, which are responsible for controlling the lamps. This document will present the fundamentals, project stages and the results obtained at the end of the work.

Keywords: Lighting Control. IoT. Microcontrollers. RS-485. Web development.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

bps	Bits por segundo
CI	Circuito integrado
IoT	Sigla em inglês para internet das coisas - *Internet of Things*
PCI	Placa de circuito impresso

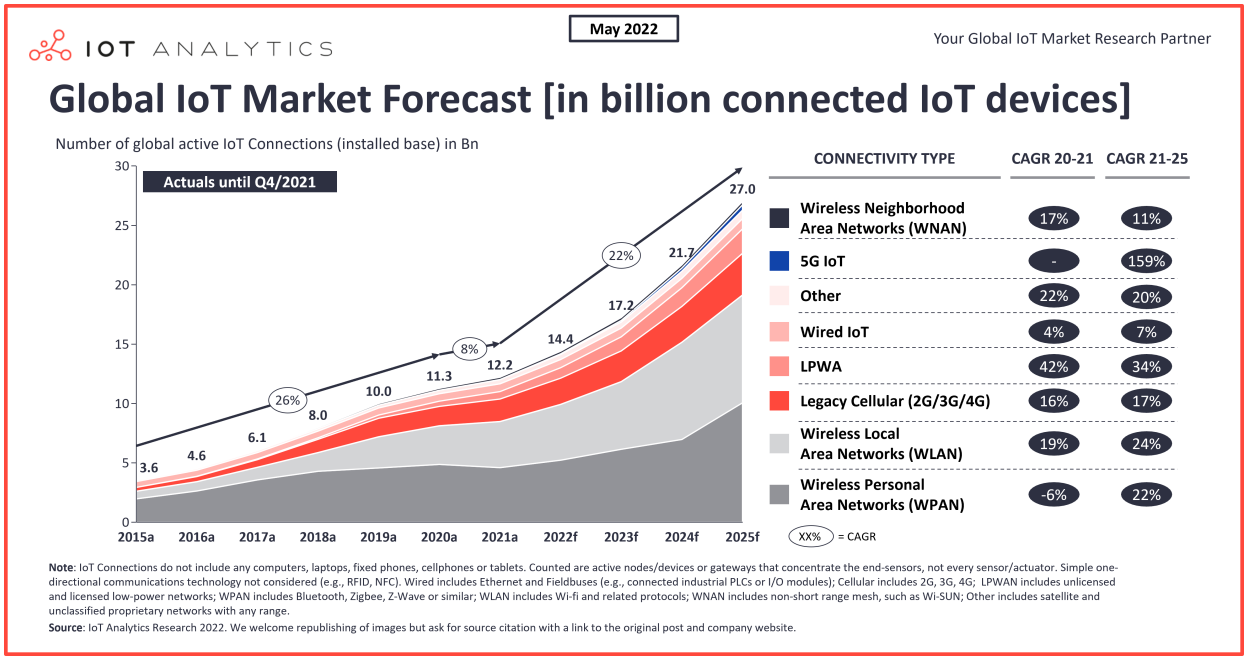
SUMÁRIO

1 INTRODUÇÃO

A cada dia fala-se mais sobre IoT, dispositivos inteligentes e afins, especialmente com o advento do 5G. Nesse contexto, surge o interesse de conectar à Internet aparelhos que antes operavam de forma isolada, como dispositivos de iluminação.

Em maio de 2022, a IoT Analytics publicou um relatório disponível em Analytics (2022), que discorre acerca do crescimento do número de dispositivos conectados. Segundo o documento, o crescimento em 2021 foi de 8%, significativamente menor que nos anos anteriores. No entanto, a expectativa é que em 2022 o crescimento seja de até 18%, embora a falta de *chips* atrapalhe a recuperação do mercado. A Figura 1 apresenta as previsões do mercado global de IoT até 2025.

Figura 1 – Previsão do mercado global de IoT



Fonte: Analytics (2022)

Ao analisar esses dados fica claro que a demanda por profissionais capazes de lidar com esse tipo de tecnologia crescerá cada vez mais, portanto é necessário que estudantes e professores dos mais variados cursos estejam se preparando para atuar nesse segmento.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver um sistema de controle remoto de iluminação residencial via Internet, que permita ao usuário monitorar, ligar e desligar lâmpadas de sua residência estando

em qualquer lugar. Além disso, o sistema deve ser flexível para que no futuro sejam adicionadas novas funcionalidades.

1.1.2 Objetivos Específicos

a) Projetar o *hardware* do sistema, aplicando o padrão RS-485 na comunicação;

b) Preparar o *layout* das PCBs;

c) Desenvolver o *software* do projeto, desde o *firmware* até a interface web;

d) Prototipar o sistema.

2 REVISÃO TEÓRICA

2.1 COMUNICAÇÃO SERIAL

Segundo Ibrahim (2014), comunicação serial é o processo de envio de dados um bit de cada vez, sequencialmente, através de um canal de comunicação ou barramento, de forma síncrona ou assíncrona. Em contraste, comunicação paralela é o envio de vários bits simultaneamente. A comunicação serial é mais simples e barata que a paralela, sendo utilizada na maioria das redes de computadores. Alguns microcontroladores possuem um bloco de *hardware* chamado *Universal Synchronous-Asynchronous Receiver–Transmitter* (USART), que implementa uma interface configurável de comunicação serial.

2.1.1 *Baud rate*

Uma das informações mais importantes em comunicação serial é o *baud rate*, que é a velocidade de transferência de dados, medida em bps. Para que a comunicação funcione corretamente, é necessário que ambos os dispositivos operem à mesma velocidade. Algumas das taxas de transmissão padrão são 1200, 2400, 4800, 9600, 57600 e 115200bps. Quanto maior a taxa, mais dados podem ser transferidos em um mesmo intervalo de tempo, mas os dispositivos e o canal são mais exigidos.

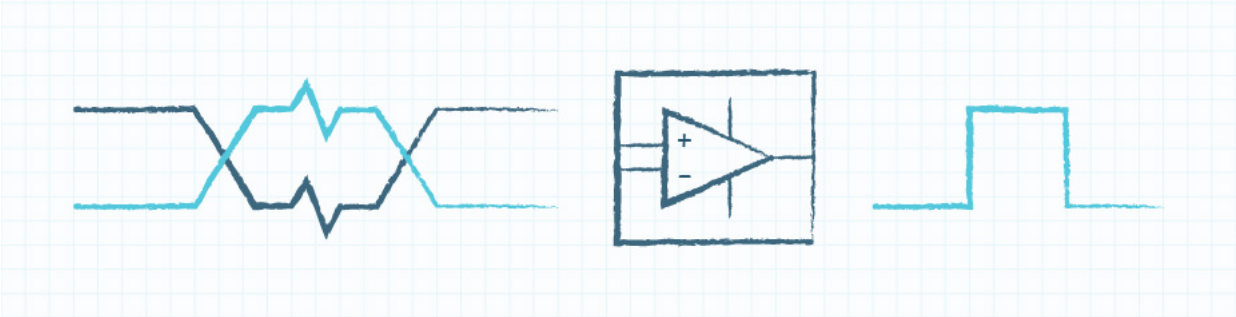
2.2 RS-485

RS-485 (*Recommendad Standart-485*) é um padrão de comunicação cabeada que especifica detalhes físicos, como níveis de tensão, velocidade e distância máxima da rede. Segundo um guia da Texas Instruments disponível em TI (2022), é amplamente utilizado em equipamentos médicos, automação industrial, embarcações, laboratórios, robótica e outras aplicações. O padrão é baseado em sinais diferenciais, permitindo a remoção de ruídos e tornarndo o RS-485 robusto a interferências, conforme visto na Figura 2.

As principais vantagens de se utilizar o padrão RS-485 são:

1. Robustez a ruídos
2. Comunicação bidirecional
3. Capacidade de criar uma rede
4. Alta velocidade de transmissão
5. Possibilidade de ter uma linha de transmissão longa

Figura 2 – Exemplo de uso do sinal diferencial para remoção de ruídos

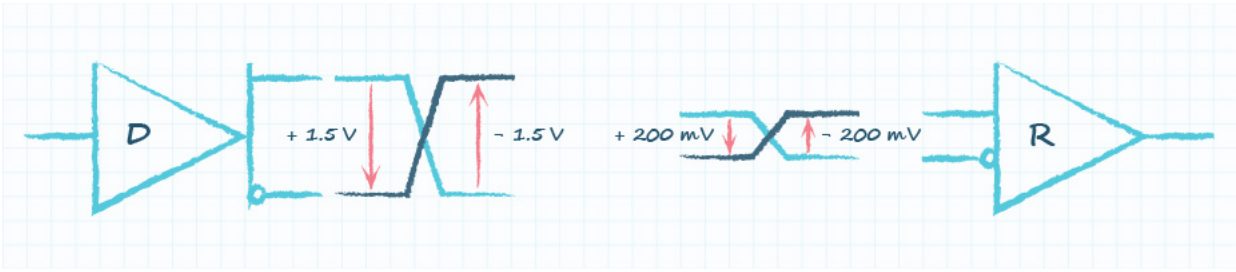


Fonte: CUI (2022)

2.2.1 Níveis de tensão

Drivers em conformidade com o padrão RS-485 fornecem uma saída diferencial de no mínimo 1.5V, enquanto os receptores em conformidade com o padrão detectam uma entrada diferencial de pelo menos 200mV (ver Figura 3). Esses níveis de tensão fornecem margem suficiente para uma transmissão de dados confiável mesmo quando há forte degradação do sinal. Essa é a principal razão pela qual o RS-485 é adequado para redes de longa distância em ambientes ruidosos.

Figura 3 – Níveis mínimos de sinal em barramentos RS-485

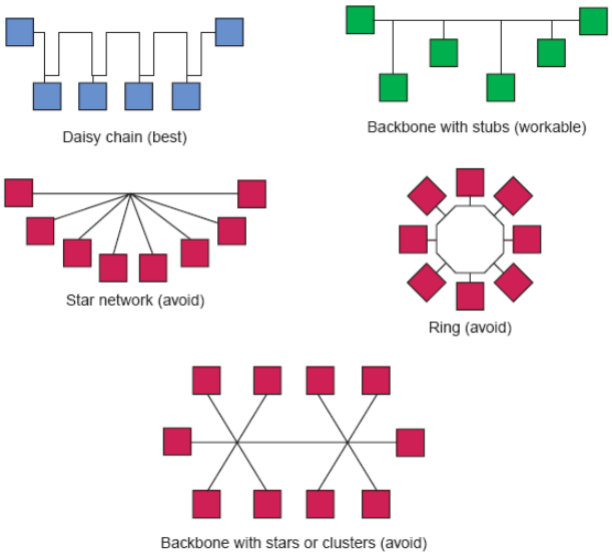


Fonte: CUI (2022)

2.2.2 Topologias

O RS-485 sugere que seus nós sejam conectados em topologias *daisy-chain* ou barramento, onde os dispositivos são conectados a um cabo principal via *stubs*. Conforme apresentado na Figura 4, é possível utilizar outras topologias, embora não seja recomendado. A rede pode ser projetada para atuar como *full-duplex* ou *half-duplex*.

Figura 4 – Topologias de rede

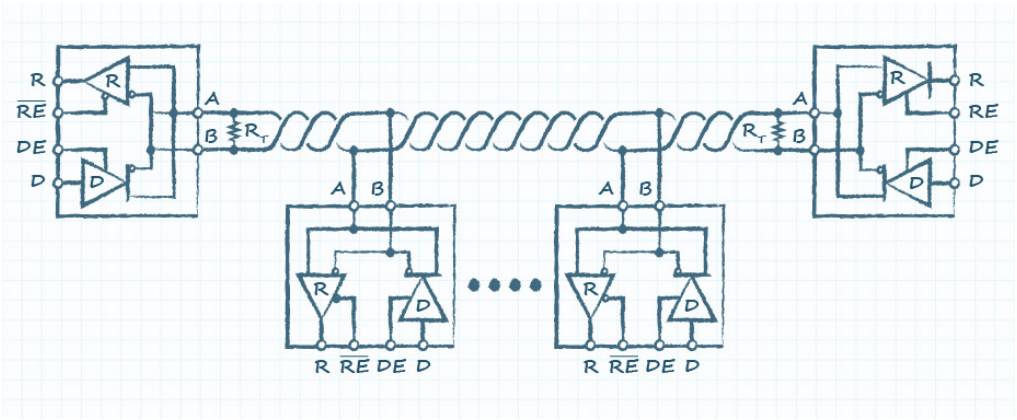


Fonte: TI (2022)

2.2.3 Gerenciamento da rede

A Figura 5 mostra uma rede RS-485 em barramento, que possibilita a comunicação bidirecional entre múltiplos dispositivos. Além disso, é possível adicionar ou remover nós da rede sem afetar os demais em funcionamento. O RS-485 é frequentemente utilizado no modo *half-duplex*, em que os dispositivos da rede se revezam na utilização do canal. O barramento deve ser gerenciado por um mestre a fim de evitar colisões de dados, que ocorrem quando múltiplos dispositivos tentam se comunicar ao mesmo tempo. O mestre controla o barramento através do envio de comandos para cada dispositivo individualmente, permitindo que ele utilize o canal por determinado período.

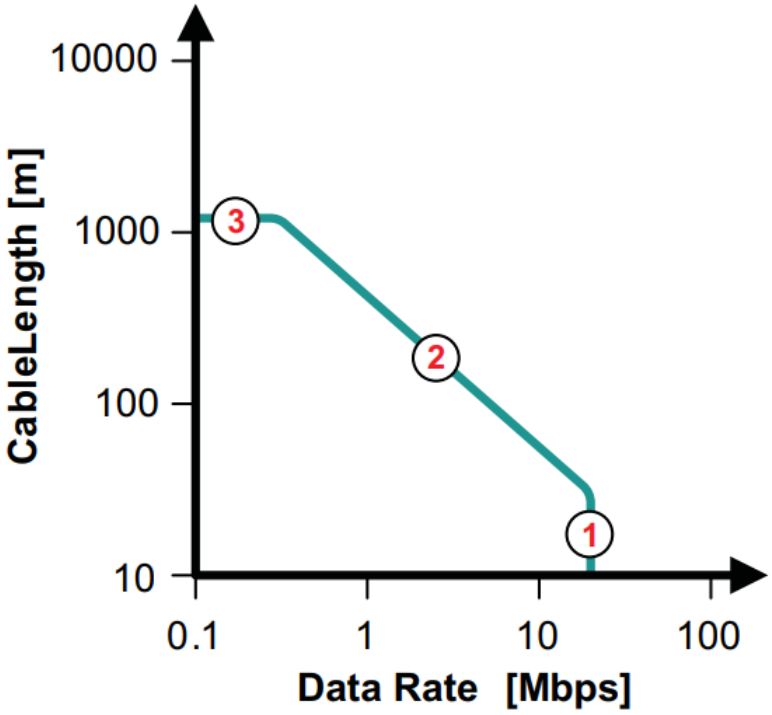
Figura 5 – Rede RS-485 em barramento



2.2.4 Velocidade de comunicação

Segundo o guia de aplicação disponível em TI (2021), a velocidade de comunicação é inversamente proporcional ao comprimento da rede e limitada a 10Mb/s (ver Figura 6), embora esse limite possa ser ultrapassado em alguns casos.

Figura 6 – Comprimento x Velocidade (bps)



Fonte: TI (2021)

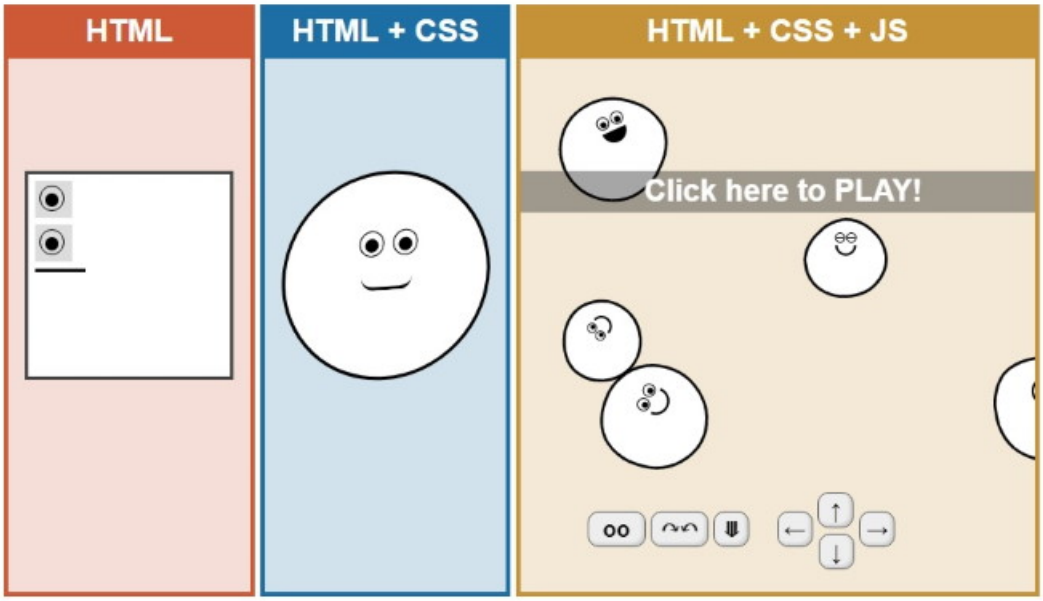
2.3 DESENVOLVIMENTO WEB

Segundo Techopedia (2022), desenvolvimento web se refere às tarefas associadas ao desenvolvimento de sites, incluindo *design*, conteúdo, escrita de *scripts*, configurações de segurança e rede, etc. Essa área abrange desde o desenvolvimento de páginas estáticas como sites corporativos, até aplicações complexas como *e-commerces* e redes sociais. O desenvolvimento web moderno é comumente dividido em duas partes: *front-end* e *back-end*

2.3.1 *Front-end*

A parte visual da web, ou seja, a interface de usuário, é chamada de *front-end*. HTML, CSS e JavaScript são linguagens fundamentais no *front-end*, responsáveis por inserir, formatar, estilizar e programar as páginas web.

Figura 7 – Papel das linguagens HTML, CSS e Javascript



Fonte: HTML-CSS-JS

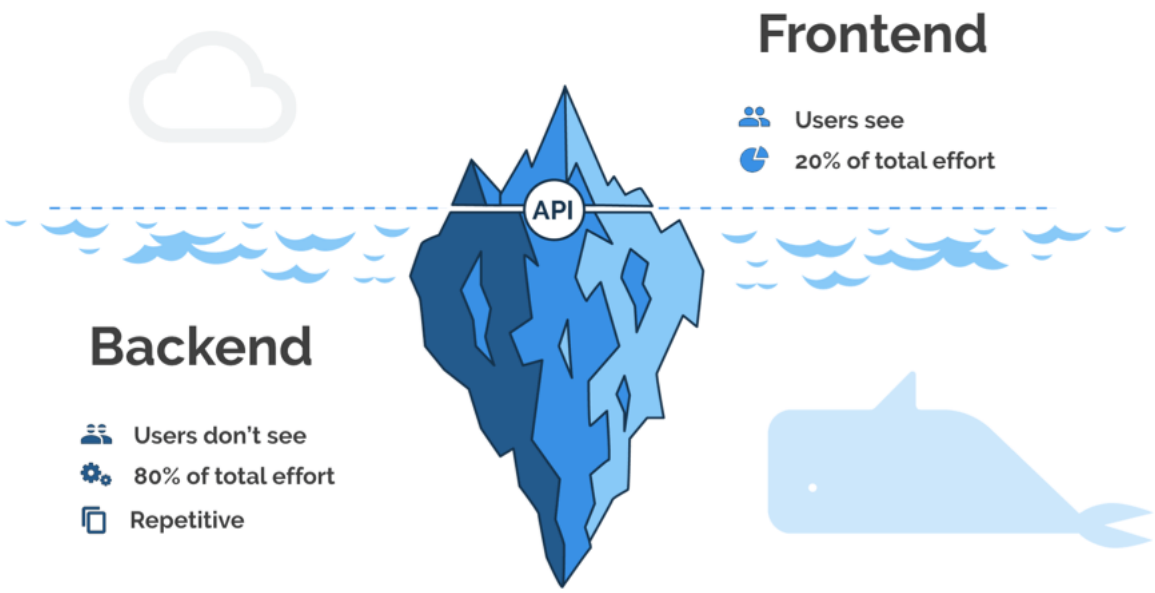
A Figura 7 permite uma melhor compreensão do papel de cada uma dessas linguagens. HTML é responsável pela inserção de conteúdos como textos, símbolos, imagens e vídeos, enquanto CSS permite que o conteúdo inserido via HTML seja estilizado das mais variadas formas. JavaScript é a linguagem de programação que pode interagir com o conteúdo da página, possibilitando o desenvolvimento de aplicações complexas.

2.3.2 *Back-end*

O *back-end* de uma aplicação é o código executado em seus servidores, em contraste ao *front-end*, que é executado nos clientes. Uma conexão direta entre *front-end* e banco de dados geraria problemas de segurança, por isso o *back-end* gerencia as conexões a bancos, disponibilizando APIs para que o *front-end* possa interagir com os dados.

As regras de negócio de uma aplicação não devem ser expostas aos seus usuários, por isso também são implementadas no *back-end*. Autenticação, geração de relatórios, processamento de pagamentos, integração com outros sistemas e mensageria são exemplos de funcionalidades implementadas no *back-end*. A Figura 8 mostra que o *back-end*, embora invisível aos usuários, é a parte mais importante e complexa de um sistema web.

Figura 8 – Comparação entre *back-end* e *front-end*



Fonte: back4app (2022)

2.3.2.1 NodeJS

Conforme visto na Seção 2.3.1, JavaScript é uma linguagem de programação pensada para *front-end* e executada em navegadores WEB como Google Chrome. No entanto, devido à sua popularidade, surgiu o interesse em levar o JavaScript para ambientes externos aos navegadores. Nesse contexto surge o NodeJS, que é um *runtime* JavaScript baseado no V8, interpretador JavaScript do Google. Graças ao NodeJS o JavaScript expandiu suas fronteiras e hoje, além de ser a linguagem dominante no *front-end*, é uma das linguagens mais utilizadas em desenvolvimento *back-end*.

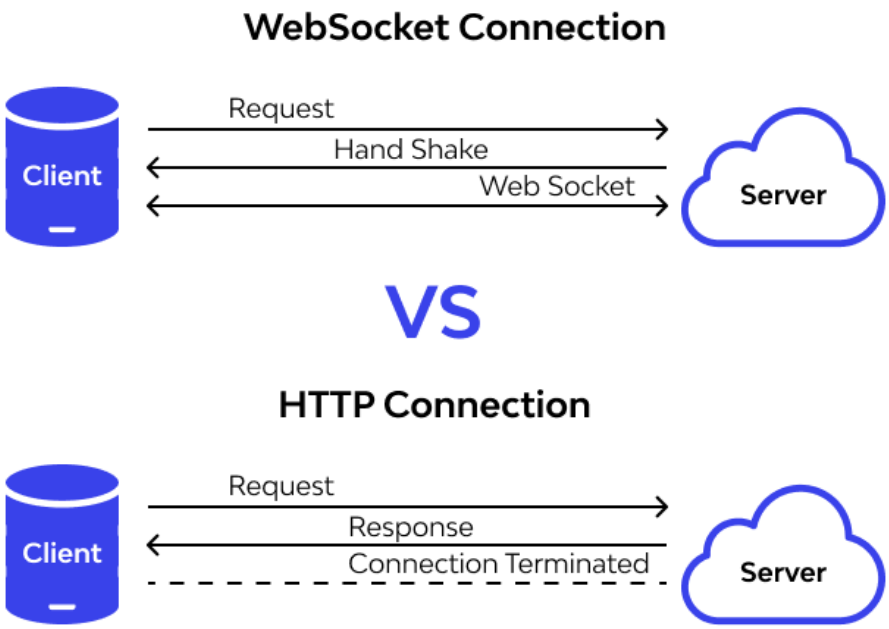
2.3.3 WebSockets

O protocolo de comunicação mais comum na internet é o HTTP. Nele, o cliente faz uma requisição ao servidor e espera uma resposta. Quando se deseja obter uma nova informação ou atualizar um dado antigo, é necessário fazer outra requisição. Para implementar sistemas que precisam de atualizações em tempo real via HTTP é utilizada uma técnica chamada *polling*, que consiste em fazer requisições ao servidor em intervalos regulares, a fim de manter as informações atualizadas. Os WebSockets surgiram para facilitar a implementação de sistemas desse tipo.

Segundo a Mozilla (2022), WebSocket é uma tecnologia avançada que permite abrir um canal de comunicação bi-direcional entre um cliente e um servidor web. Em HTTP, apenas o cliente pode fazer requisições ao servidor, mas WebSockets permitem

que o servidor também envie mensagens ao cliente. Implementar sistemas de tempo real via WebSockets é vantajoso, pois não é necessário utilizar *polling*, já que o próprio servidor pode comunicar ao cliente qualquer atualização ocorrida nos dados, reduzindo o tráfego de informações pela internet.

Figura 9 – WebSockets VS HTTP

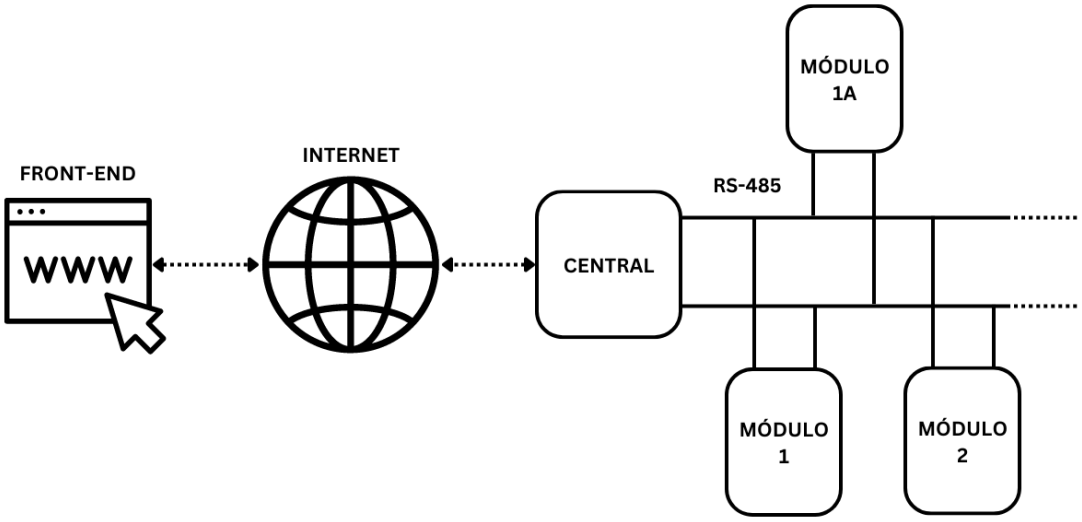


Fonte: Wallarm (2022)

3 DESENVOLVIMENTO

3.1 ARQUITETURA DO SISTEMA

Figura 10 – Diagrama de blocos de arquitetura do sistema



Fonte: O autor

A Figura 10 apresenta o diagrama de blocos utilizado como base para o desenvolvimento do protótipo do sistema, que, objetivando simplicidade, está conectado a apenas quatro lâmpadas, mas não é limitado a esse número. As seções a seguir detalharão cada um dos blocos.

3.2 *FRONT-END*

O *front-end* da aplicação é uma página web (código disponível no Apêndice B) desenvolvida utilizando as tecnologias abordadas na Seção 2.3.1. Há um botão para controlar cada lâmpada, bem como botões para ligar ou desligar todas as luzes (ver Figura 11). Além disso, as cores dos botões variam de acordo com o estado atual de cada lâmpada, sendo que verde indica luz acesa, enquanto vermelho indica luz apagada. Essas cores são atualizadas a cada mensagem recebida da central.

Clicar nos botões dispara mensagens que devem ser enviadas à central via Internet. Isso ocorre através de um *back-end* simples escrito em NodeJS (código disponível no Apêndice A), cuja única função é redirecionar as mensagens enviadas pelo *front-end* à central e vice-versa. Tanto a página web quanto a central do sistema estão conectadas à Internet, mas não conseguem se comunicar diretamente, por

isso ambas se conectam a um servidor cujo endereço é conhecido e ele atua como intermediário da comunicação.

Figura 11 – Capturas de tela do *Front-end* do sistema

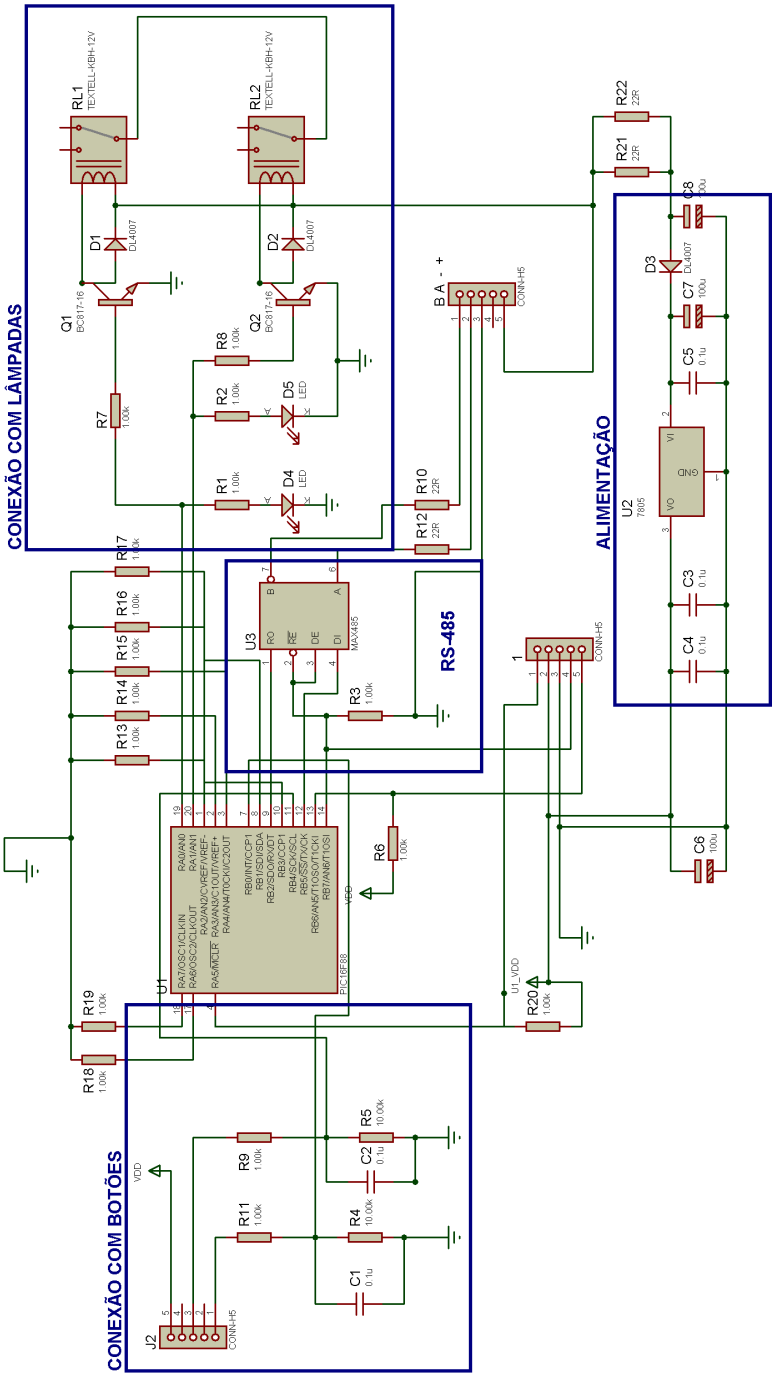
Fonte: O autor

3.3 MÓDULOS

Os módulos são blocos de *hardware* que se conectam às lâmpadas do ambiente, controlando-as de acordo com os comandos recebidos pela rede e também atuando como interruptores. Há duas categorias de módulos: principais e auxiliares. Os módulos principais se conectam diretamente às lâmpadas (cada módulo pode se conectar a dois pontos), sendo nomeados sequencialmente como M1, M2, M3, ..., Mn. Por sua vez, os módulos auxiliares atuam como interruptores paralelos, recebendo o nome do seu módulo principal mais a letra "A" (M1A, M2A, M3A, ..., MnA).

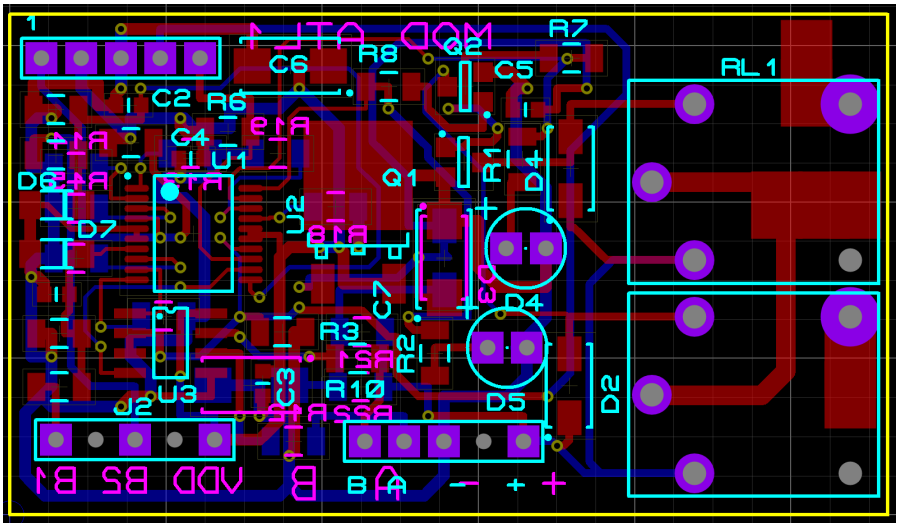
O esquemático do circuito dos módulos, cujo principal componente é um microcontrolador PIC16F88 (código disponível no Apêndice D), é apresentado na Figura 12, enquanto as Figuras 13 e 14 mostram o *layout* da PCI de um módulo e o circuito já montado, respectivamente. O esquemático e a PCI dos módulos foram projetados no Proteus, *software* para criação de projetos eletrônicos.

Figura 12 – Esquemático dos módulos



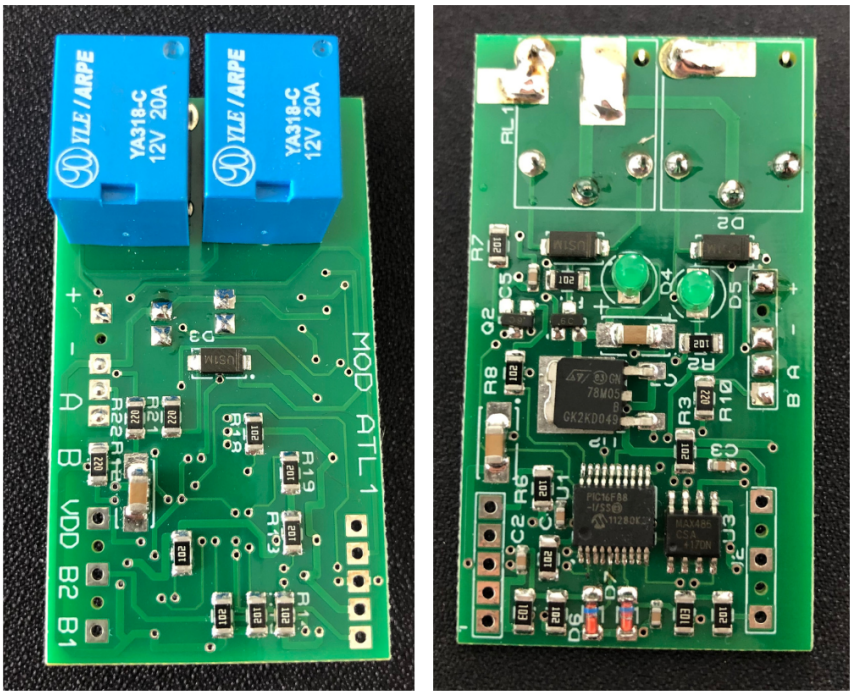
Fonte: O autor

Figura 13 – *Layout* da PCI



Fonte: O autor

Figura 14 – Circuito montado na PCI

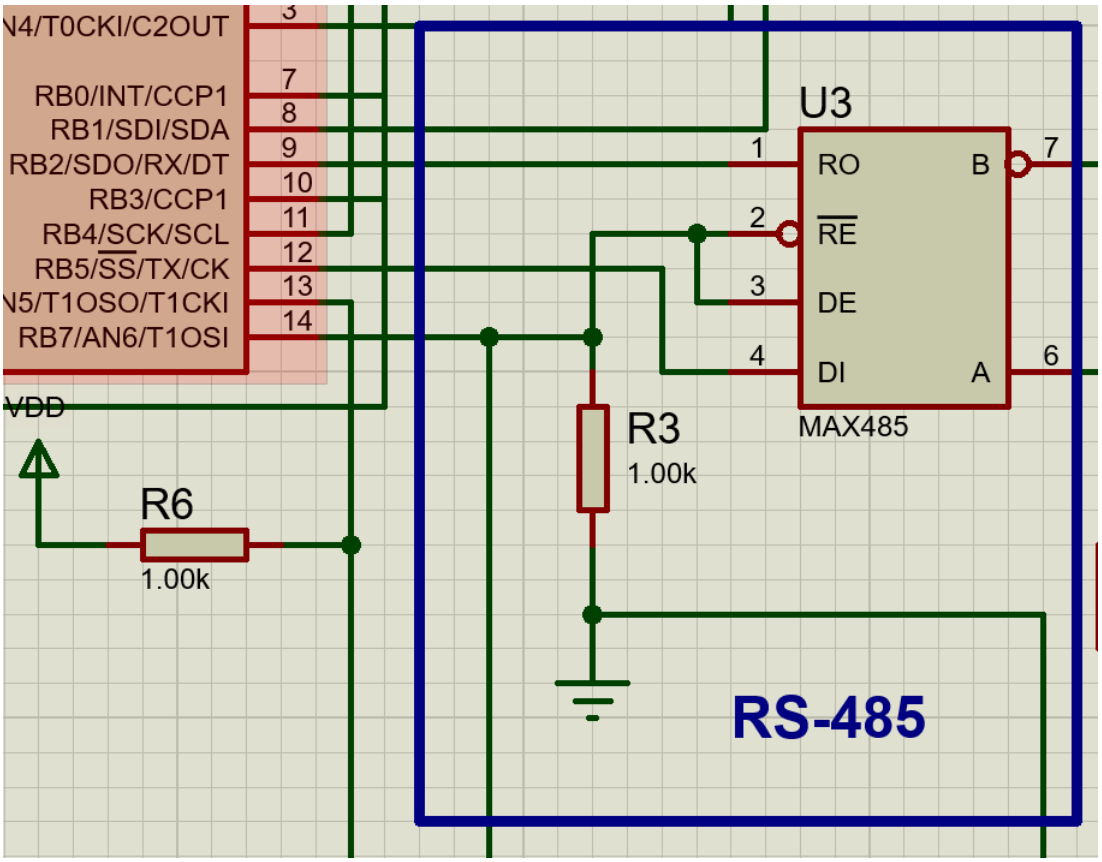


Fonte: O autor

3.3.1 Padrão RS-485

A escolha do padrão RS-485 foi baseada em duas características principais: robustez da comunicação e simplicidade de implementação. Para adequar a comunicação ao padrão, cada módulo possui um transceptor MAX-485 da Maxim Integrated, CI dedicado para comunicação em redes RS-485. A conexão entre o PIC e o MAX-485 é feita através de três pinos: TX, RX e CONTROLE. Os pinos de TX e RX são utilizados na comunicação de sinais transmitidos e recebidos, enquanto o pino de controle é responsável por definir se o MAX-485 está em modo de transmissão (CONTROLE = 1) ou recepção (CONTROLE = 0). A comunicação entre os módulos na rede ocorre através dos canais diferenciais do MAX-485, chamados de A e B.

Figura 15 – Conexão do MAX-485

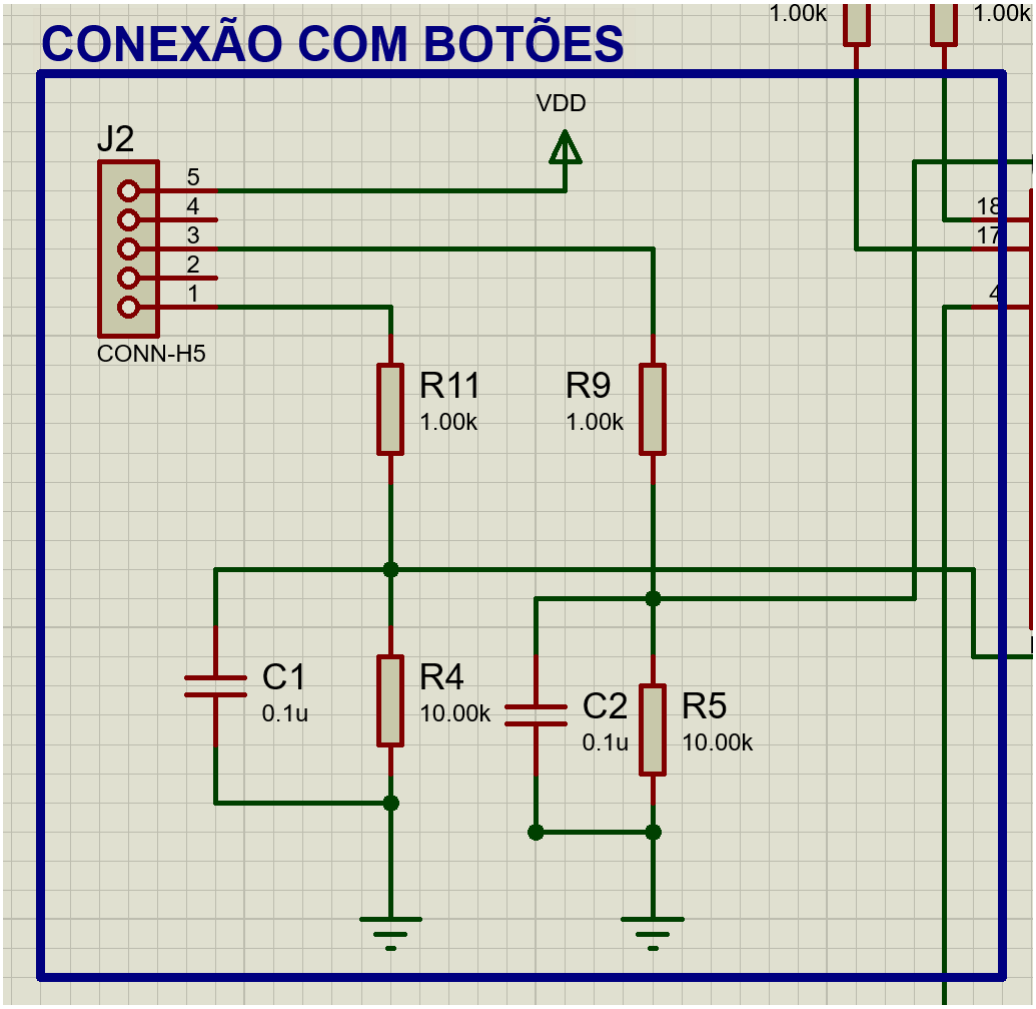


Fonte: O autor

3.3.2 Conexão com botões

Cada módulo é conectado a dois botões, que atuam como interruptores das lâmpadas. Para lidar com os efeitos de *bouncing*, cada botão está conectado a um divisor de tensão com filtro RC passa-baixas (ver Figura 16), além de haver uma segunda camada de proteção no *firmware* do microcontrolador.

Figura 16 – Divisor de tensão com filtro RC

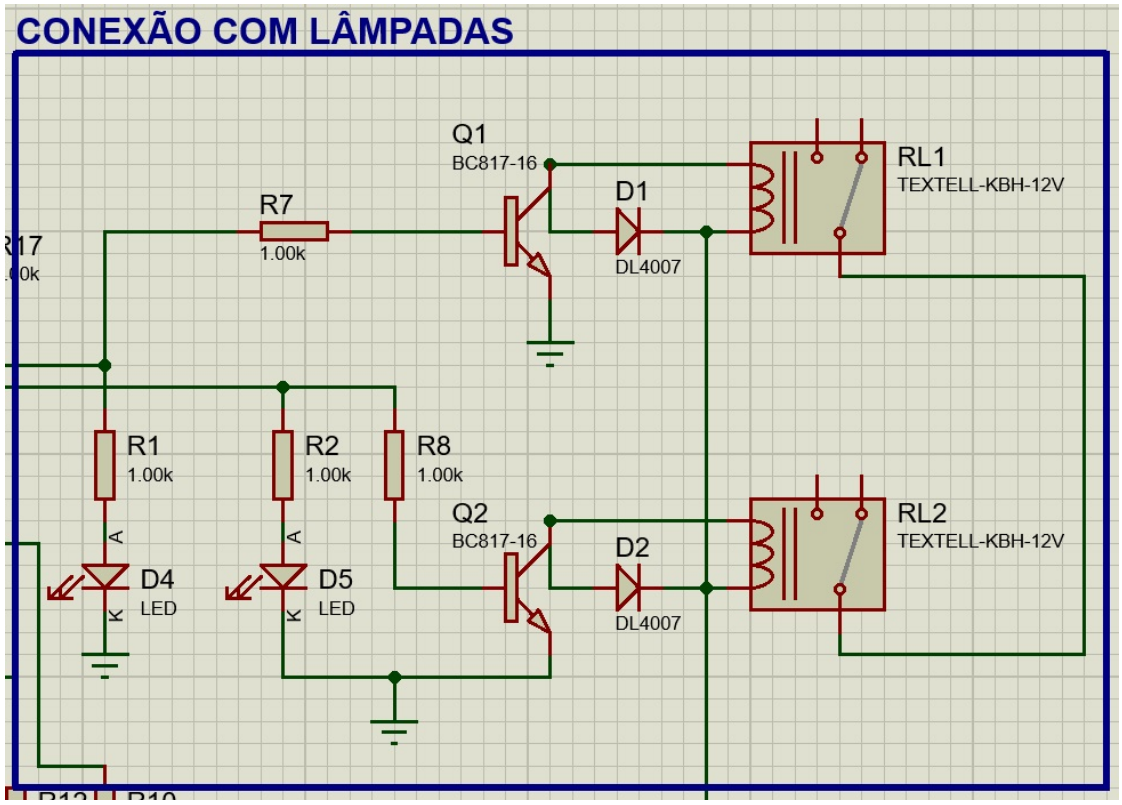


Fonte: O autor

3.3.3 Conexão com lâmpadas

Cada módulo é conectado a duas lâmpadas, cuja alimentação é 110V ou 220V AC. O chaveamento das lâmpadas é implementado com relés, que devem ser acionados pelo microcontrolador. Um dos terminais do relé está conectado à tensão de 12V que alimenta o circuito, enquanto o outro está associado a um transistor NPN conectado ao terra. Quando o transistor conduz, há uma tensão de 12V sobre os terminais do relé, acionando o dispositivo e permitindo que a alimentação AC chegue à lâmpada. Quando o transistor não conduz, não há corrente no relé, portanto o dispositivo é desativado e a alimentação da lâmpada é interrompida. O controle dos transistores é realizado pelo PIC.

Figura 17 – Circuito de chaveamento da iluminação

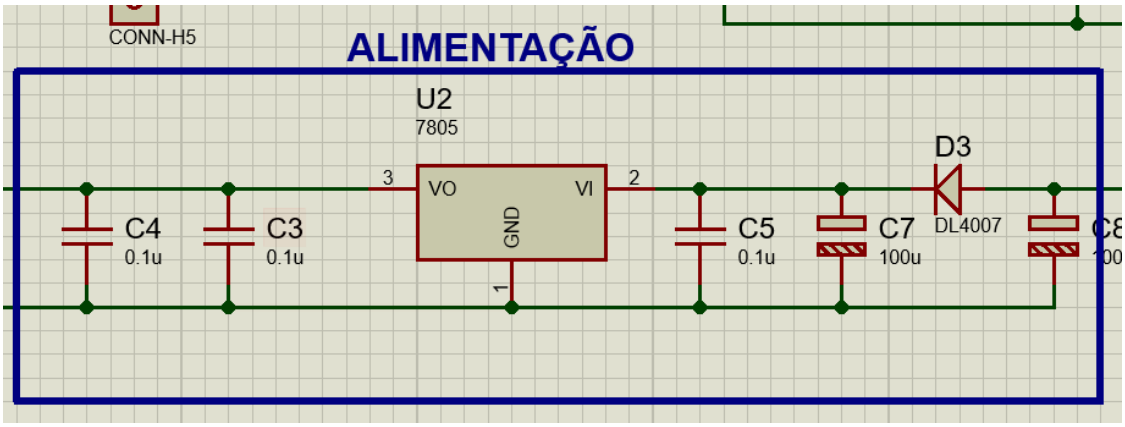


Fonte: O autor

3.3.4 Alimentação

O circuito é alimentado por 12V DC, necessários para acionar os relés, e possui um regulador de tensão que gera 5V para alimentar o microcontrolador PIC e o MAX-485.

Figura 18 – Conexão do regulador de 5V



Fonte: O autor

3.4 CENTRAL

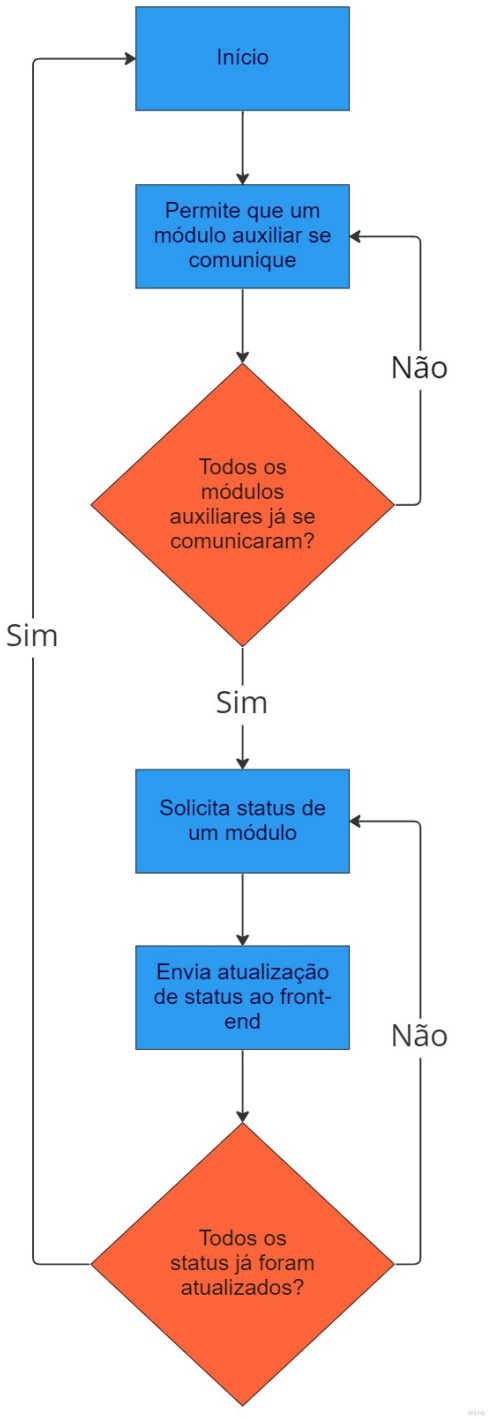
O bloco denominado "central" foi implementado utilizando a linguagem Python e uma Raspberry Pi 4, orquestrando o funcionamento de todo sistema através de comunicação com o *front-end* e gerenciamento do barramento RS-485. A Figura 19 apresenta o fluxograma de trabalho da central, um loop infinito (a implementação em código está disponível no Apêndice C).

O início do fluxograma mostra que a central realiza periodicamente varreduras nos módulos auxiliares, permitindo que se comuniquem. Os módulos auxiliares não se conectam diretamente às lâmpadas, pois atuam como interruptores paralelos, enviando comandos a seu módulo principal. Por padrão, nenhum dispositivo pode se comunicar através do barramento a menos que a central permita, portanto, quando um botão de um módulo auxiliar é pressionado, o dispositivo aguarda até receber permissão da central para se comunicar e, ao receber permissão, envia a seu módulo principal o comando para alterar o estado da lâmpada correspondente.

Após gerenciar a comunicação dos módulos auxiliares, a central envia atualizações de *status* ao *front-end*, mantendo o estado de cada lâmpada atualizado na página. Para isso, a central solicita que cada módulo principal lhe envie o estado de suas lâmpadas, aguarda a resposta e envia a informação ao *front-end*. Depois de realizar estes passos para cada módulo principal, a central retorna ao início do *loop*.

Durante a execução das etapas descritas no fluxograma, o código pode ser interrompido por comandos enviados pelo *front-end*. Quando isso ocorre, a central executa o comando recebido e depois retorna ao fluxo padrão de execução.

Figura 19 – Fluxograma de trabalho da central



Fonte: O autor

3.5 PROTOCOLO DE COMUNICAÇÃO

A comunicação entre central e módulos ocorre através do protocolo representado na Figura 20. A função de cada *byte* do protocolo é descrita a seguir:

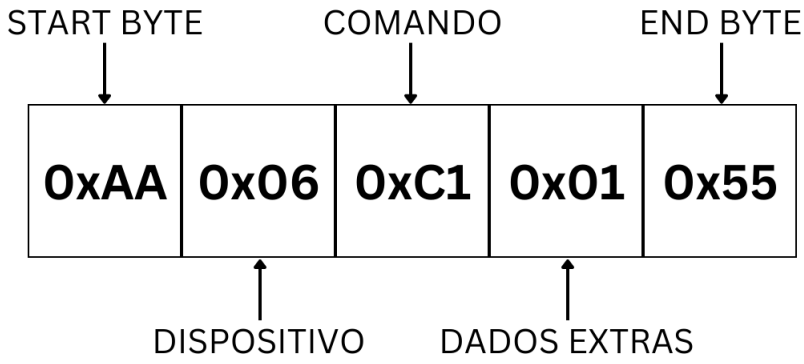
a) *Start byte* e *end byte*: Atuam como delimitadores das mensagens enviadas pela rede;

b) Dispositivo: Endereço do dispositivo alvo da mensagem, podendo variar entre 0 (0x00) e 254 (0xFE), pois o valor 255 (0xFF) é o endereço de *broadcast*;

c) Comando: Valor que especifica qual comando deve ser executado pelo dispositivo alvo da mensagem. A Tabela 1 contém alguns exemplos;

d) Dados extras: *Byte* utilizado para transferir dados, quando necessário. Por exemplo: quando um módulo envia uma mensagem de atualização de *status* à central, o estado das lâmpadas é transportado por esse *byte*.

Figura 20 – Protocolo implementado



Fonte: O autor

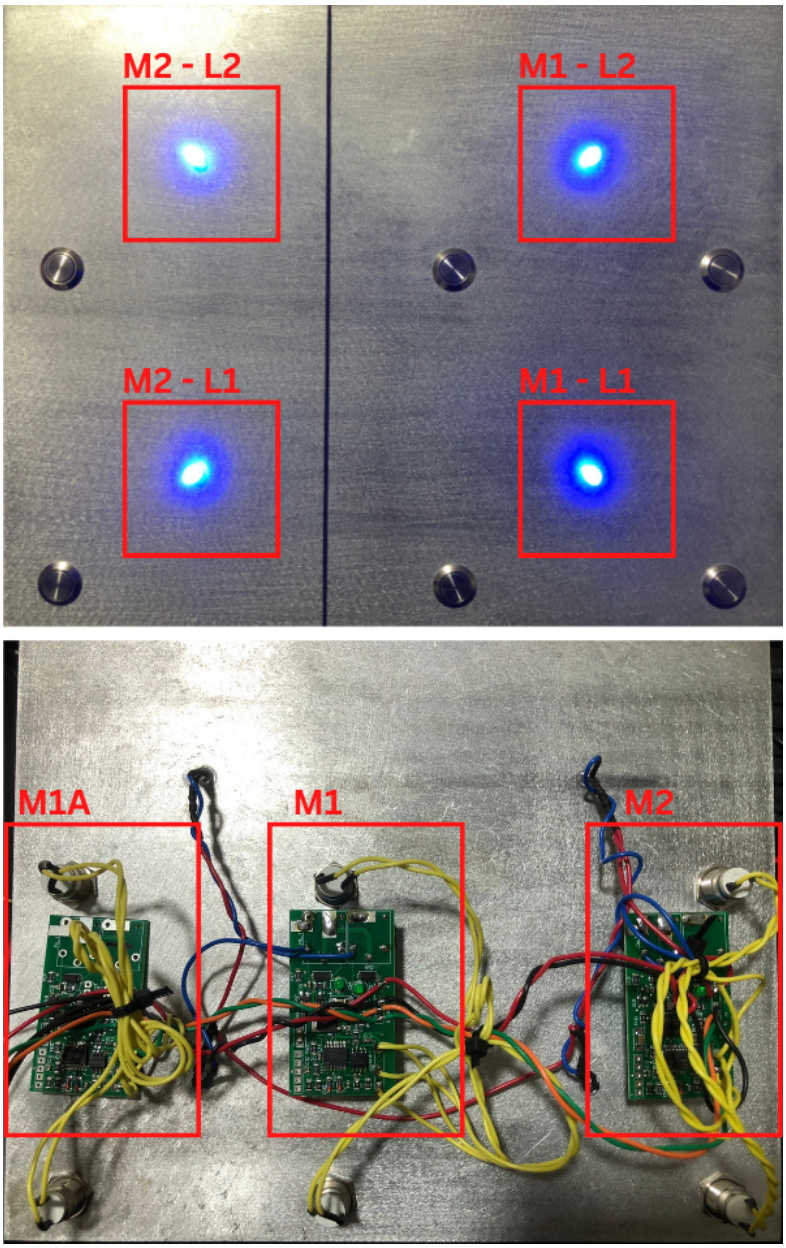
Tabela 1 – Alguns dos comandos implementados no protocolo

Comando	**Código**
Trocar estado da lâmpada 1	0xA0
Trocar estado da lâmpada 2	0xA1
Acender todas as lâmpadas	0xB0
Apagar todas as lâmpadas	0xB1
Atualizar status	0xC2

3.6 PROTÓTIPO

A Figura 21 apresenta o protótipo do sistema. Os módulos principais M1 e M2 controlam os LEDs, que simulam as lâmpadas de uma residência, enquanto o módulo auxiliar M1A implementa os interruptores paralelos do módulo M1 e, embora não apareça na figura, há uma Raspberry atuando como central do sistema. Todas as funcionalidades do projeto foram testadas e funcionaram como previsto, então é possível controlar as luzes pelos botões ou pela interface web e seu estado é atualizado em tempo real na página.

Figura 21 – Protótipo do sistema



Fonte: O autor

4 CONCLUSÃO

Os objetivos gerais e específicos do trabalho foram atingidos, pois foi desenvolvido um protótipo funcional de um sistema de controle de iluminação residencial via Internet, incluindo projeto de *hardware*, *firmware*, *layout* de PCBs e interface web. O aspecto mais positivo do projeto é sua capacidade de expansão, pois é possível projetar e conectar à rede mais módulos com diferentes funcionalidades, tornando o sistema mais completo e útil. Ademais, um sistema conectado à Internet pode ser integrado a serviços de terceiros, ampliando suas possibilidades.

4.1 TRABALHOS FUTUROS

A principal melhoria a ser realizada em uma futura revisão é a substituição de comunicação com fio por comunicação sem fio. O uso de RS-485 trouxe simplicidade de implementação e robustez ao sistema, mas a necessidade de interligar os módulos através de cabos dificulta a implementação do projeto em residências já construídas. Além disso, do ponto de vista comercial, um sistema sem fio é mais atrativo aos consumidores.

REFERÊNCIAS

BACK4APP. **Top 20 Backend Tools List**. [*S.l.*], 2022. Disponível em: <https://blog.back4app.com/backend-tools-list/>. Acesso em: 2 nov. 2022. Citado na p. 21.

CUI DEVICES. **RS-485 Serial Interface Explained**. [*S.l.*], 2022. Disponível em: <https://www.cuidevices.com/blog/rs-485-serial-interface-explained>. Acesso em: 4 out. 2022. Citado na p. 17.

IBRAHIM, Dogan. **PIC Microcontroller Projects in C**. 2. ed. [*S.l.*: *s.n.*], 2014. Disponível em: <https://www.sciencedirect.com/book/9780080999241/pic-microcontroller-projects-in-c>. Acesso em: 15 nov. 2022. Citado na p. 16.

IOT ANALYTICS. **State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally**. [*S.l.*], 2022. Disponível em: <https://iot-analytics.com/number-connected-iot-devices/>. Acesso em: 27 out. 2022. Citado na p. 14.

JR., Louis E. Frenzel. **Principles of Electronic Communication Systems**. 4. ed. [*S.l.*: *s.n.*], 2022. P. 378. Disponível em: <https://physicaeducator.files.wordpress.com/2018/03/principles-of-electronic-communication-system-by-luies.pdf>. Acesso em: 7 dez. 2022.

MAXIM INTEGRATED. **MAX481/MAX483/MAX485/ MAX487–MAX491/MAX1487 Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers**. [*S.l.*], 2022. Disponível em: <https://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf>. Acesso em: 7 dez. 2022.

MICROCHIP. **PIC16F87/88 18/20/28-Pin Enhanced Flash MCUs with nanoWatt Technology**. [*S.l.*], 2022. Disponível em: <http://ww1.microchip.com/downloads/en/devicedoc/30487d.pdf>. Acesso em: 7 dez. 2022.

MOZILLA. **The WebSocket API (WebSockets)**. [*S.l.*], 2022. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API>. Acesso em: 2 nov. 2022. Citado na p. 21.

OPENJS FOUNDATION. **About Node.js**. [*S.l.*], 2022. Disponível em: <https://nodejs.org/en/about/>. Acesso em: 7 dez. 2022.

TECHOPEDIA. **Web Development**. [*S.l.*], 2022. Disponível em: <https://www.techopedia.com/definition/23889/web-development>. Acesso em: 2 nov. 2022. Citado na p. 19.

TEXAS INSTRUMENTS. **The RS-485 Design Guide**. [*S.l.*], 2021. Disponível em: <https://www.ti.com/lit/an/slla272d/slla272d.pdf?ts=1664971674469>. Acesso em: 5 out. 2022. Citado na p. 19.

TEXAS INSTRUMENTS. **The RS-485 Reference Guide**. [*S.l.*], 2022. Disponível em: <https://www.compel.ru/wordpress/wp-content/uploads/2019/05/slyt484a.pdf>. Acesso em: 13 nov. 2022. Citado nas pp. 16, 18.

WALLARM. **A Simple Explanation Of What A WebSocket Is**. [*S.l.*], 2022. Disponível em: <https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is>. Acesso em: 2 nov. 2022. Citado na p. 22.

Apêndices

APÊNDICE A – CÓDIGO DO SERVIDOR WEB

```
const http = require('http');
const express = require('express');
const { Server } = require('socket.io');

const app = express();
const server = http.createServer(app);
const io = new Server(server, {
  cors: {
    origin: '*',
  },
});

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/public/index.html');
});

io.on('connection', (socket) => {
  socket.broadcast.emit('message', 'conected');

  socket.on('update_state', (info) => {
    console.log(info);
    socket.broadcast.emit('update_state', info);
  });

  socket.on('command', (command) => {
    console.log(socket.id, command);
    socket.broadcast.emit('command', command);
  });
});

const port = process.env.PORT || 3000;
server.listen(port, () => {
  console.log(`listening on port ${port}`);
});
```

APÊNDICE B – CÓDIGO DA PÁGINA WEB

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <script src="https://cdn.tailwindcss.com"></script>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.1.1/css/all.min.css"
      integrity="sha512-KfkfwYDsLkIlwQp6LFnl8zNdLGxu9YAA1QvwINks4PhcElQSvqcyVLLD9aMhXd13uQjoXtEKNosOWaZqXgel0g=="
      crossorigin="anonymous" referrerpolicy="no-referrer" />
    <title>Client</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.5.1/socket.io.js"
      integrity="sha512-9mpsATI0KClwt+xVZfbcf2lJ8IFBAwsubJ6mI3rtULwyM3fBmQFzj0It4tGqxLOGQwGfJdk/G+fANnxfq9/cew=="
      crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <style>
      .btn {
        margin: 3px;
        width: 64px;
        height: 40px;
        border: 1px solid;
        border-radius: 10px;
        border-color: white;
      }

      .btn-grow {
        transition: all 0.05s ease-in-out;
      }

      .btn-grow:hover {
        transform: scale(1.03);
      }
    </style>
  </head>

  <body class="w-screen h-screen bg-gray-900 text-white flex items-center justify-center">
    <div class="flex flex-col h-fit">
      <div class="">
        <label class="m-3">M1 - L1:</label>
        <button id="02_L1" class="btn btn-grow" onclick="onOffClick(event)">
          <i class="fa-solid fa-power-off"></i>
        </button>
      </div>
      <div class="">
        <label class="m-3">M1 - L2:</label>
```

```
      <button id="02_L2" class="btn btn-grow" onclick="onOffClick(event)">
        <i class="fa-solid fa-power-off"></i>
      </button>
    </div>
    <div class="">
      <label class="m-3">M2 - L1:</label>
      <button id="06_L1" class="btn btn-grow" onclick="onOffClick(event)">
        <i class="fa-solid fa-power-off"></i>
      </button>
    </div>
    <div class="">
      <label class="m-3">M2 - L2:</label>
      <button id="06_L2" class="btn btn-grow" onclick="onOffClick(event)">
        <i class="fa-solid fa-power-off"></i>
      </button>
    </div>
    <div class="mt-5 ml-2">
      <button class="btn btn-grow" onclick="allOn()">
        All ON
      </button>
      <button class="btn btn-grow" onclick="allOff()">
        All OFF
      </button>
    </div>
  </div>
  <script>
    const socket = io();

    socket.on('connect_error', (error) => {
      console.log(error);
    });

    socket.on('update_state', (info) => {
      const L1 = document.getElementById(`0${info['module']}_L1`);
      const L2 = document.getElementById(`0${info['module']}_L2`);
      switch (info['state']) {
        case 0:
          setColors(L1, 'red');
          setColors(L2, 'red');
          break;
        case 1:
          setColors(L1, 'green');
          setColors(L2, 'red');
          break;
        case 2:
```

```
            setColors(L1, 'red');
            setColors(L2, 'green');
            break;
          case 3:
            setColors(L1, 'green');
            setColors(L2, 'green');
            break;
        }
      })

      function setColors(element, color) {
        element.style.color = color
        element.style.borderColor = color
      }

      function onOffClick(event) {
        let target = event.target.nodeName == 'BUTTON' ? event.target : event.target.parentElement;
        [mod, com] = target.id.split('_');
        data = { module: Number(mod), command: `SWITCH_${com}` };
        socket.emit('command', data);
      }

      function allOn() {
        data = { module: 255, command: `ALL_ON` };
        socket.emit('command', data);
      }

      function allOff() {
        data = { module: 255, command: `ALL_OFF` };
        socket.emit('command', data);
      }
    </script>
  </body>
</html>
```

APÊNDICE C – CÓDIGO DA CENTRAL

```
import serial
import socketio

from time import sleep

from devices import *
from commands import *

ser=serial.Serial('/dev/serial0', 125000)

DEVICE = CENTRAL
MODULOS = [M1, M2]

BUFFER_SIZE = 5
END_BYTE = 0x55
START_BYTE = 0xAA



def send_message(message):
    ser.reset_input_buffer()
    for byte in message:
        data = bytes(chr(byte), encoding='latin-1')
        ser.write(data)

def read_message():
    msg = [int(hex(ord(c)), 16) for c in file.read()]
    for i in range(len(msg) - BUFFER_SIZE + 1):
        sub_message = msg[i:i + BUFFER_SIZE]
        if sub_message[0] == START_BYTE and sub_message[4] == END_BYTE and sub_message[1] == DEVICE:
            return sub_message
    return []



if __name__ == '__main__':
    file = open('/dev/serial0', encoding='latin-1')

    sio = socketio.Client()
    try:
        sio.connect('https://lights-control.herokuapp.com/')
    except socketio.exceptions.ConnectionError as e:
        print(e)
        exit()

    @sio.on('*')
```

```
def catch_all(event, data):
    if event == 'command':
        send_message([START_BYTE, data['module'], eval(data['command']), 0x00, END_BYTE])


while True:
    for mod in MODULOS:
        send_message([START_BYTE, mod + 1, TRANSMIT, 0x00, END_BYTE])
        sleep(0.05)

    for mod in MODULOS:
        send_message([START_BYTE, mod, UPDATE_STATUS, 0x00, END_BYTE])
        sleep(0.05)
        msg = read_message()
        if msg:
            sio.emit('update_state', {'module': mod, 'state': msg[3]})
```

APÊNDICE D – CÓDIGO DOS MÓDULOS

```
#include <stdbool.h>

// Inclui lista de dispositivos
#include "dispositivos.dat"

// Define dispositivo
const char device = M1;

// Constantes
#define      end_byte                0x55
#define      start_byte              0xAA
#define      debounce_time           0x02
#define      input_buffer_size       0x02
#define      output_buffer_size      0x05

// Comandos do Protocolo
#define      switch_L1               0xA0
#define      switch_L2               0xA1
#define      switch_L12              0xA2
#define      all_on                  0xB0
#define      all_off                 0xB1
#define      transmit                0xC0
#define      update_status           0xC2

// Entradas
#define      BUT1                    PORTB.B0
#define      BUT2                    PORTB.B4

// Saídas
#define      L1                      PORTA.B0
#define      L2                      PORTA.B1
#define      control                 PORTB.B7

// Funções gerais
void handle_RX(void);
void PIC_setup(void);
void clear_input_buffer(void);
void transmit_protocol(void);
void transmit_command(char ip, char cmd, char dado);

// GPRs
char i;
char Lbackup;
char Lstatus;
```

```
char aux1_AUSART;
char aux2_AUSART;
char AUSART_flags;
char timer0_counter;
char timer2_counter;
char input_buffer[input_buffer_size];
char output_buffer[output_buffer_size];

// Bits
sbit device_flag         at AUSART_flags.B3;
sbit protocol_RX         at AUSART_flags.B0;
sbit end_byte_flag       at AUSART_flags.B1;
sbit start_byte_flag     at AUSART_flags.B2;

// Tratador de interrupçõeses de alta prioridade
void interrupt(){
     //Interrupção da AUSART
     if(PIR1.RCIF){
         handle_RX();
         PIR1.RCIF = 0;
     }

     //Interrupção externa
     if(INTCON.INT0IF){
         L1 = !L1;
         INTCON.INT0IF = 0;
     }

     //Interrupção por mudança de estado
     if(INTCON.RBIF){
         if(BUT2 == 1){
             T2CON.TMR2ON = 1;
             timer2_counter = 0;
         }
         INTCON.RBIF = 0;
     }

     //Interrupção do TIMER2
     if(PIR1.TMR2IF){
         timer2_counter++;
         if(timer2_counter == debounce_time){
             T2CON.TMR2ON = 0;
             if(BUT2 == 1) L2 = !L2;
         }
         PIR1.TMR2IF = 0;
```

```
    }
}

// Tratador de interrupções de baixa prioridade
void interrupt_low(){

}

// Programa principal
void main(){
    PIC_setup();                                //Configura o PIC
    while(1){
        // Lida com instruções recebidas pela RS-485
        if(protocol_RX){
            switch(input_buffer[0]){
                case switch_L1:
                    L1 = !L1;
                    break;
                case switch_L2:
                    L2 = !L2;
                    break;
                case switch_L12:
                    L1 = !L1;
                    L2 = !L2;
                    break;
                case all_on:
                    L1 = 1;
                    L2 = 1;
                    break;
                case all_off:
                    L1 = 0;
                    L2 = 0;
                    break;
                case update_status:
                    transmit_command(central, update_status, PORTA&0x03);
                    break;
                default:
                    break;
            }
            protocol_RX = 0;
        }
    }
}

// Configura o PIC
```

```
void PIC_setup(){
     // Configuração do oscilador interno
     OSCCON = 0x70;                              //Fosc = 8MHz

     // Desabilita o watchdog timer
     WDTCON = 0x00;

     // Configuração de PORTs
     ANSEL  = 0x00;                              //ANSEL = B'00000000'        (0 = digital; 1 = analog)
     TRISA  = 0xFC;                              //TRISA = B'11111100'        (0 = output; 1 = input)
     TRISB  = 0x5F;                              //TRISB = B'01011111'        (0 = output; 1 = input)
     PORTA  = 0x00;
     PORTB  = 0x00;

     // Configuração de interrupções
     INTCON = 0xC0;                              //Habilita interrupções gerais e de periféricos

     // Habilita interrupção externa
     INTCON.INT0IE = 1;

     // Habilita interrupção por mudança de estado
     INTCON.RBIE = 1;

     // Configuração da AUSART (RS-485)
     PIE1     = 0x20;                            //Habilita interrupção RX da AUSART2
     TXSTA    = 0x26;                            //Transmissão assíncrona de 8 bits
     SPBRG    = 0x03;                            //Baud Rate = 57600
     RCSTA    = 0x90;                            //Habilita porta serial
     aux1_AUSART = 0;
     aux2_AUSART = 0;

     output_buffer[0] = start_byte;
     output_buffer[output_buffer_size-1] = end_byte;

     // Configuração do TIMER2
     PR2          = 0xFA;
     T2CON        = 0x26;
     PIE1.TMR2IE     = 1;
     T2CON.TMR2ON    = 0;

     delay_ms(100);                              //Delay para estabilização de periféricos
}

// Lida com dados recebidos pela AUSART (RS-485)
void handle_RX(){
```

```
     aux2_AUSART = RCREG;
     if(start_byte_flag){
          if(device_flag){
               if(end_byte_flag){
                    aux1_AUSART = 0;
                    end_byte_flag = 0;
                    start_byte_flag = 0;
                    device_flag = 0;

                    if(aux2_AUSART == end_byte) protocol_RX = 1;
                    else clear_input_buffer();
               }
               else{
                    input_buffer[aux1_AUSART] = aux2_AUSART;
                    aux1_AUSART++;
                    if(aux1_AUSART == input_buffer_size) end_byte_flag = 1;
               }
          }
          else{
               if(aux2_AUSART == device || aux2_AUSART == broadcast) device_flag = 1;
               else start_byte_flag = 0;
          }
     }
     else{
          if(aux2_AUSART == start_byte) start_byte_flag = 1;
     }
}

// Transmite os bytes do protocolo de comunicação
void transmit_protocol(){
     for(i=0; i<output_buffer_size; i++){
          TXREG = output_buffer[i];
          while(!TXSTA.TRMT);
     }
}

// Transmite um comando pela rede
void transmit_command(char ip, char cmd, char dado){
     output_buffer[1] = ip;
     output_buffer[2] = cmd;
     output_buffer[3] = dado;

     control = 1;
     transmit_protocol();
     control = 0;
```

```
}

// Limpa buffer de entrada da AUSART
void clear_input_buffer(){
     for(i=0; i<input_buffer_size; i++){
          input_buffer[i] = 0;
     }
}
```