Universidade Federal
do Rio de Janeiro
Escola Politécnica

IMPLEMENTAÇÃO DE UM SISTEMA DE AUTOMAÇÃO RESIDENCIAL UTILIZANDO A PLATAFORMA ARDUINO E OS MÓDULOS DE RÁDIO FREQUÊNCIA XBEE

Jonas Karst Simões

Projeto de Graduação apresentado ao Curso de Engenharia de Controle e Automação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Lilian Kawakami Carvalho

Rio de Janeiro
Agosto de 2018

IMPLEMENTAÇÃO DE UM SISTEMA DE AUTOMAÇÃO RESIDENCIAL UTILIZANDO A PLATAFORMA ARDUINO E OS MÓDULOS DE RÁDIO FREQUÊNCIA XBEE

Jonas Karst Simões

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE AUTOMAÇÃO.

Examinado por:

Prof. Lilian Kawakami Carvalho, D.Sc.

Prof. Alessandro Jacoud Peixoto, D.Sc.

Prof. Cláudio Miceli de Farias, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
AGOSTO DE 2018

Karst Simões, Jonas

Implementação de um sistema de automação residencial utilizando a plataforma Arduino e os módulos de rádio frequência XBee/Jonas Karst Simões. – Rio de Janeiro: UFRJ/ Escola Politécnica, 2018.

XI, 98 p.: il.; 29, 7cm.

Orientador: Lilian Kawakami Carvalho

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia de Controle e Automação, 2018.

Referências Bibliográficas: p. 47 – 48.

1. Automação Residencial. 2. Arduino. 3. XBee. I. Kawakami Carvalho, Lilian. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Título.

Agradecimentos

Gostaria de agradecer em primeiro lugar aos meus pais, José Carlos e Soraya. Não poderia ser diferente. Sem todo o incentivo e apoio deles, a conclusão deste projeto e do curso de Engenharia de Controle e Automação não seriam possíveis. Vocês são meus pilares e exemplos de pessoas para mim. Sou imensamente grato pela educação que vocês me proporcionaram. Muito obrigado! Eu amo vocês!

Gostaria de agradecer também a todos os amigos que tive o prazer de conhecer ao longo do curso, que tornaram essa jornada muito menos complicada e mais prazerosa. Vocês tem um lugar especial junto de mim. Um agradecimento especial ao Eduardo Brandão, minha eterna dupla de todos os trabalhos!

Não poderia deixar de agradecer a todos os membros da Equipe Minerva Aerodesign que tive o prazer de trabalhar, que me mostraram o verdadeiro significado da palavra Equipe, e que fizeram o curso de engenharia ser muito mais interessante.

Por fim, a última pessoa que fez parte da minha caminhada, minha orientadora, Lilian Kawakami Carvalho. Muito obrigado por acreditar no meu projeto e por toda a atenção dedicada durante esse período. Sua ajuda foi muito imporante para a realização deste trabalho.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Automação.

IMPLEMENTAÇÃO DE UM SISTEMA DE AUTOMAÇÃO RESIDENCIAL UTILIZANDO A PLATAFORMA ARDUINO E OS MÓDULOS DE RÁDIO FREQUÊNCIA XBEE

Jonas Karst Simões

Agosto/2018

Orientador: Lilian Kawakami Carvalho

Curso: Engenharia de Controle e Automação

O avanço tecnológico dos celulares e a evolução das redes sem fio revolucionou a maneira como as pessoas se relacionam entre si e também com o ambiente a sua volta, possibilitando que as pessoas interajam com o mundo através de dispositivos eletrônicos. O surgimento da internet das coisas, permitiu avanço em diversas áreas da tecnologia, como por exemplo a domótica, também conhecida como automação residencial . Neste trabalho, implementamos um sistema de automação residencial que é controlado através de aplicativo em smartphone, que foi desenvolvido através da ferramenta MIT App Inventor. Para o desenvolvimento do sistema, foram utilizados a plataforma Arduino e os módulos de comunicação por radio frequência XBee. Este sistema foi projetado para acionar atuadores relé e leds infravermelhos, capazes de controlar iluminação e eletrônicos que fazem uso de controle remoto. O desenvolvimento do sistema foi descrito passo a passo para permitir a replicação deste projeto por terceiros.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer.

IMPLEMENTATION OF A HOME AUTOMATION SYSTEM USING ARDUINO AND THE RADIO FREQUENCY XBEE MODULES

Jonas Karst Simões

August/2018

Advisor: Lilian Kawakami Carvalho

Course: Automation and Control Engineering

The technological advancement of mobile phones and the evolution of wireless networks has revolutionized the way people relate to each other and also to the environment around them, enabling people to interact with the world through electronic devices. The emergence of the Internet of Things (IoT) has allowed advances in several areas of technology, such as domotics, also known as home automation. In this work, we implemented a home automation system that is controlled through a smartphone application, which was developed through the MIT App Inventor tool. For the development of the system, the Arduino platform and XBee radio frequency communication modules were used. This system is designed to drive relay actuators and infrared leds, capable of controlling lighting and electronics that make use of remote control. The development of the system has been described step by step to allow the replication of this project by third parties.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Na última década vivemos a evolução da internet das coisas (IoT) graças aos avanços tecnológicos na área das redes sem fio e da telefonia móvel. A internet das coisas é uma rede de dispositivos físicos que possuem componentes eletrônicos, sensores, atuadores, e conexão à internet, permitindo assim a comunicação entre esses dispositivos e o compartilhamento de dados entre eles. A ideia por trás da internet das coisas é extender a conexão à internet além de dispositivos comuns como tablets, celulares e computadores, para quaisquer dispositivos tradicionais que não são conectados à internet, como: veículos, eletrodomésticos, tubulações, equipamentos médicos. A integração destes dispositivos em uma rede, permite o monitoramento, controle e automação de processos em diversas aplicações: transporte, industria, saúde, construção, casas inteligentes, dentre outros. Em meio a esses avanços, o homem se tornou de capaz de interagir com outras pessoas e com o mundo a sua volta através de dispositivos eletrônicos portáteis, melhorando a sua qualidade de vida. Um dos campos mais beneficiados com esses recentes avanços foi o da automação residencial, também conhecida como domótica. A domótica, da junção Domus "casa"com Imotique "automático, é definida como a integração de mecanismos automáticos em um ambiente, tornando o cotidiano das pessoas mais simples, solucionando necessidades de comunicação, conforto e segurança [1]. Ela permite o uso de dispositivos com o objetivo de automatizar ou complementar as rotinas e tarefas de uma residência, permitindo o controle de temperatura ambiente, iluminação e acionamento de outros eletrodomésticos. Na área da automação residencial, porém, nem sempre todos os dispositivos são conectados à internet. Existem diferentes arquiteturas que podem ser utilizadas para a implementação de um sistema de automação residencial. Uma arquitetura horizontal, onde os dispositivos são todos conectados à internet, como na definição de internet das coisas, ou uma rede centralizada, em que um dispositivo principal é conectado à internet e distribui os comandos através de outro protocolo de comunicação, como será visto nesse trabalho.

A motivação para o desenvolvimento deste trabalho surgiu dentro de casa, a partir de pequenos conflitos e situações encontradas no dia a dia. Pra começar, só no meu quarto são cinco controles remotos para controlar ar condicionado, ventilador de teto, televisão, net e porta retrato digital, sendo que alguns dos dispositivos são controlados apenas pelo controle remoto, impossibilitando assim controlar esses aparelhos quando os controles desaparecem. Outro fator que me motivou a desenvolver este trabalho foi a preguiça. Quem nunca deitou na cama pra dormir e depois lembrou que havia deixado algo ligado em outro cômodo, ou até mesmo esqueceu de apagar a luz, cujo interruptor fica longe da cama? Por fim, e principal motivo para este projeto: meus pais. Não sei quantas vezes ao dia eles reclamam de que alguma coisa na casa está ligada, ou que eu ou meus irmãos esquecemos algo ligado, e temos que parar imediatamente o que estamos fazendo para ir desligar. Após vinte e cinco anos de reclamações eu falei irônicamente que iria fazer um aplicativo para meu pai poder controlar tudo da casa sem sair do quarto dele, já que ele adorava brincar de fiscal. Como eu estava a ponto de começar o projeto final, decidi então por este tema.

O objetivo deste trabalho, portanto, é desenvolver um sistema de acionamento remoto de atuadores, a partir de um aplicativo no celular, utilizando uma tecnologia mais barata, de fácil manuseio e aprendizado, para que outras pessoas pudessem usufruir dos benefícios deste trabalho no futuro, mesmo não tendo muita experiência com projetos eletrônicos. Portanto, desde o início decidiu-se pela implementação dessa idéia utilizando a plataforma Arduino, uma plataforma de prototipagem eletrônica de hardware livre, tão conhecido por ser um componente open source e que incentiva a troca de conhecimento constante entre os membros de sua comunidade. Para realizar a comunicação do arduino com os atuadores periféricos, optou-se por utilizar os módulos de rádio frequência XBee, pois possuem bom alcance, vasta opção de modos de operação, baixo consumo e alta confiabilidade.

Os componentes utilizados na realização deste trabalho estão descritos no capítulo 2, que detalha o funcionamento e a configuração de todos os elementos utilizados no sistema: Arduino, XBee e MIT App Inventor. Em seguida, no capítulo 3, está explicado a metodologia proposta e como é feita a integração entre os componentes do sistema. Por fim, no capítulo 4, são apresentados os resultados obtidos com a realização deste projeto, bem como as dificuldades encontradas durante a sua realização. Ao longo deste trabalho há a presença de tabelas e paragrafos contendo palavras em inglês, por se tratarem de termos técnicos retirados do manual de um dos componentes utilizados. A tradução livre destes termos poderia resultar no não entendimento do leitor ao tentar reproduzir este projeto.

Capítulo 2

Descrição dos componentes

Nesse capitulo são apresentados os componentes utilizados na implementação do sistema de automação residencial deste trabalho: plataforma arduino, módulos de comunicação por rádio frequência XBee e o ambiente de desenvolvimento de aplicativos MIT App Inventor. Serão descritos as especificações técnicas, seus componentes físicos, e os seus ambientes de desenvolvimento, focando, principalmente, nos recursos de cada componente que serão utilizados para implementação do sistema.

2.1 Arduino

O Arduino é uma plataforma eletrônica de código aberto desenvolvido com o objetivo de oferecer uma maneira mais simples e fácil de utilizar e integrar software e hardware. Essa plataforma combina as versáteis placas eletrônicas e o software de desenvolvimento do Arduino (IDE), em que os códigos são escritos utilizando a linguagem de programação C++. Existem diferentes modelos de placas Arduino, e todas elas são compostas por um micro controlador, diversas portas digitais e analógicas, pinos de alimentação de entrada e saída, e comunicação serial. Além da simplicidade de integração entre software e hardware, essa plataforma se destaca pelo baixo custo, pela versatilidade de utilização em vários sistemas operacionais, e principalmente pela ampla comunidade de usuários que se criou, devido à sua ideologia de código aberto, incentivando a troca de experiências e informações entre os membros da comunidade, facilitando aprendizado através de exemplos e bibliotecas de códigos disponíveis na internet. Somado a isso, o custo das placas Arduino é bem pequeno se comparado a outras plataformas um pouco mais sofisticadas, sendo mais um atrativo para a escolha dessa plataforma. Uma placa arduino uno custa vinte reais no varejo, valor quatro vezes menor que uma placa Raspberry pi e dez vezes menos que uma placa raspberry pi 3, por exemplo.

A placa escolhida para esta aplicação foi o Arduino Uno R3, que é baseada no

micro controlador ATmega328P, e está representada na figura 2.1. Pode-se observar que a placa é composta 14 portas digitais que podem ser configurados como entrada ou saída (portas 0 a 13 na parte superior da imagem), das quais seis funcionam como saída analógica (portas 3, 5, 6, 9, 10 e 11) e duas para comunicação serial (pino 0 de entrada e pino 1 de saída). Ela possui ainda 6 portas de entradas analógicas (A0 até A5 no canto inferior direito), dois pinos de alimentação de saída, uma de 3.3V e outra de 5V, um pino de alimentação de entrada (5V), além de um pino de reset. A placa conta também com uma conexão USB para comunicação serial, um botão de reset físico, uma entrada para fonte externa, e um header ICSP (In-circuit serial programming), que serve para comunicação com periféricos, como módulos externos e Shields [1], através do protocolo SPI (serial peripheral interface). O header ICSP possui dois pinos de alimentação (VCC e terra), um pino de controle (reset), e três pinos para conexão SPI: MOSI, pino que envia dados ao periférico, MISO, que recebe dados do periférico, e SCK, que controla o pulso/clock da conexão.

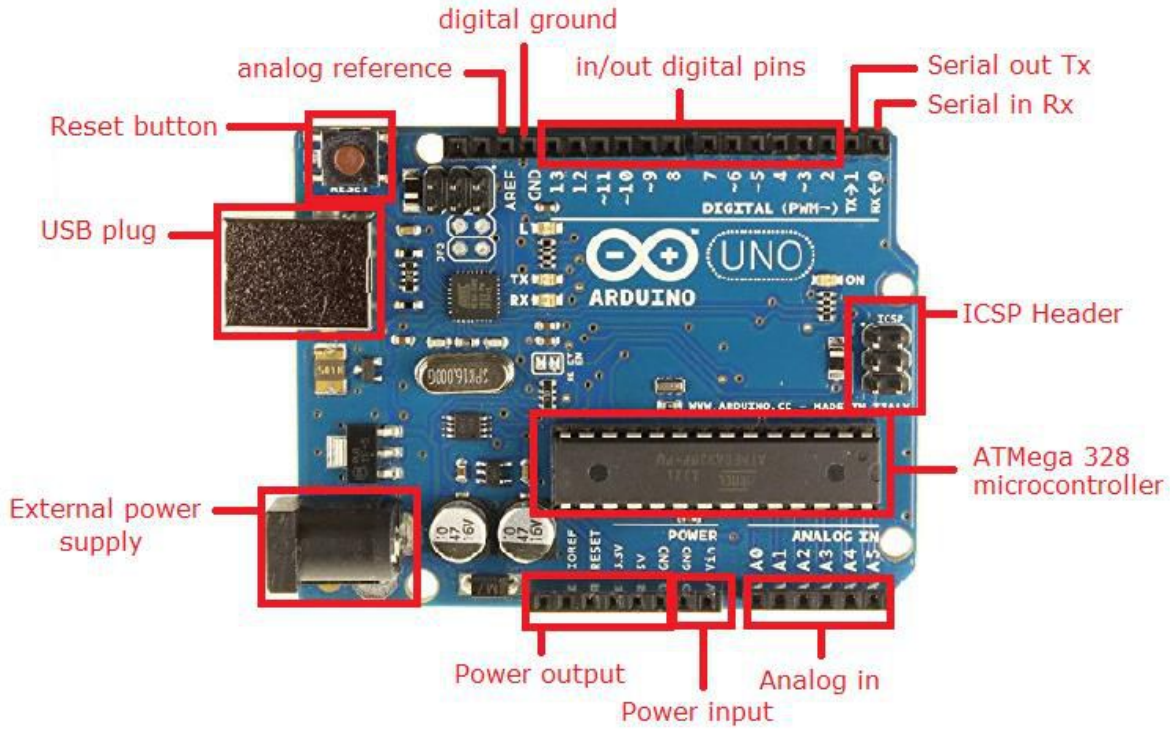

Figura 2.1: Pinagem da placa Arduino Uno R3

A tabela 2.1 mostra as especificações técnicas da placa Arduino Uno R3. Ela opera com cinco volts de tensão, podendo ser alimentada com tensão entre seis e vinte volts. O consumo de corrente nas portas digitais é baixo, de apenas 20

[1] placa de circuito impresso com conectores que se encaixam na parte superior de uma placa arduino (Citar wikipedia)

Tabela 2.1: Especificações técnicas do Arduino Uno R3

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12 V
Input Voltage (limit)	6-20 V
Digital I/O Pins	14 (6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3 pins	50 mA
Flash Memory	32 KB (ATmega328P) 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Lenght	68.6 mm
Width	53,4 mm
Weight	25 g

mA por porta, exceto a porta de alimentação com 3.3V que fornece 50 mA. A memória flash do micro controlador é de 32 KB, considerada baixa se comparada a outros micro controladores. Em aplicações de monitoramento de sensores e controle de atuadores, porém, essa configuração é suficiente, e foi utilizada devido aos seguintes pontos: simplicidade, ampla comunidade interativa e capacidade de se agregar funcionalidades ao micro controlador através de módulos externos e Shields. A comunidade do Arduino incentiva a troca de experiência entre os usuários, e possui fóruns online oficiais e não oficiais para troca de conteúdo, ideias de projetos, bibliotecas de programação, dentre outras informações. Somado a isso, as placas arduino são muito intuitivas de serem utilizadas, elas funcionam através da configuração das portas digitais como entrada ou saída, para realizar uma infinidade de aplicações distintas, através da integração dos mais variados de módulos ou Shields.

Os módulos externos e Shields são placas eletrônicas, sensores, ou atuadores, que quando conectados ao Arduino agregam alguma nova funcionalidade ao mesmo. A grande diferença entre eles é a maneira com que são conectados ao Arduino, como será visto a seguir.

Na figura 2.2, temos o diagrama de conexão entre o Arduino Uno R3 e um módulo relé de dois canais. Pode-se perceber que a conexão é realizada por meio de fios interligando a placa e o módulo relé. Este recebe o positivo (5 volts) e negativo (terra) da alimentação do arduino, além de dois fios de controle dos relés, um para cada canal.

Os Shields, em contrapartida, são projetados para serem acoplados em cima do Arduino, como visto na figura 2.3, em que temos uma placa Arduino Uno R3 junto a um XBee Shield, possibilitando ao Arduino utilizar comunicação por rádio frequência com outros módulos XBee. Os shields podem ainda ser empilhados, quando vários são utilizados, caso não haja conflito entre as portas utilizadas por cada um, ou incompatibilidade física entre eles.

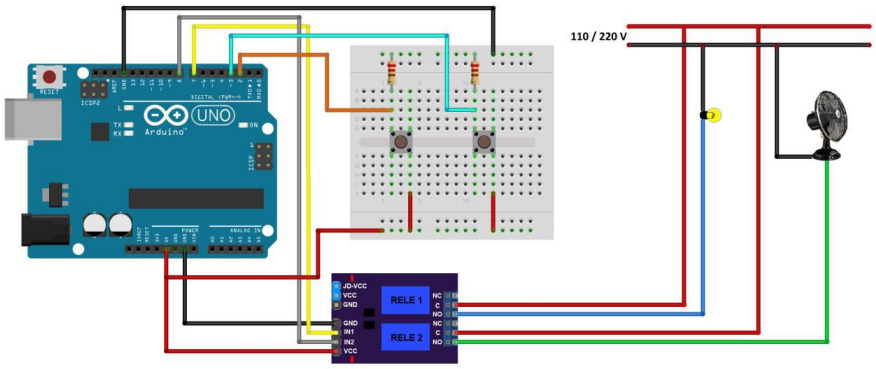

Figura 2.2: Módulo relé de dois canais para arduino

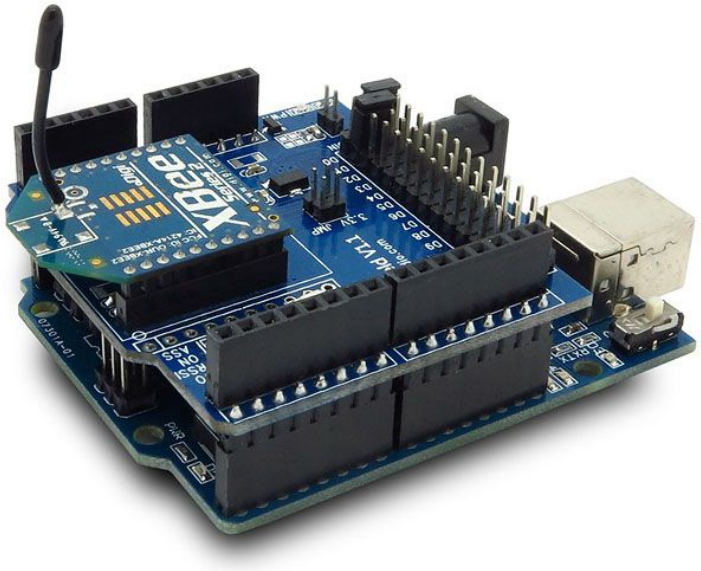

Figura 2.3: Shield XBee para arduino

Para a aplicação deste trabalho, que utiliza a internet ou rede local para acionar os atuadores, foi utilizado o Ethernet Shield W5100, que possibilita o Arduino se conectar à internet através de um cabo de rede ethernet. Esse shield, que está representado na figura 2.4, é baseado no chip WIZnet ethernet 5100 que fornece acesso à rede (IP) através dos protocolos TCP ou UDP. Ele é compatível com o Arduino Uno e Arduino Mega, e possui um slot para cartão de memória micro SD para armazenar dados, que podem inclusive ser utilizados na rede. A comunicação do Arduino Uno R3 com o chip WIZnet ethernet 5100 e com o cartão micro SD ocorre pelos pinos 11, 12 e 13, que espelham em paralelo os pinos MOSI, MISO e SCK do header ICSP. A entrada digital 10 é usada para selecionar o W5100, e o pino 4 para o cartão micro SD, impossibilitando o uso destes para outros propósitos. O shield ainda conta com um botão de reset, que reinicia tanto o shield quanto a placa Arduino a que ele esteja acoplado, como uma série de leds que indicadores de status como: se o shield está energizado, se há conexão à rede, ocorrência de envio e recepção de dados, velocidade da conexão de rede e indicador de colisão na rede. A tabela 2.2 relaciona os leds a suas respectivas funções.

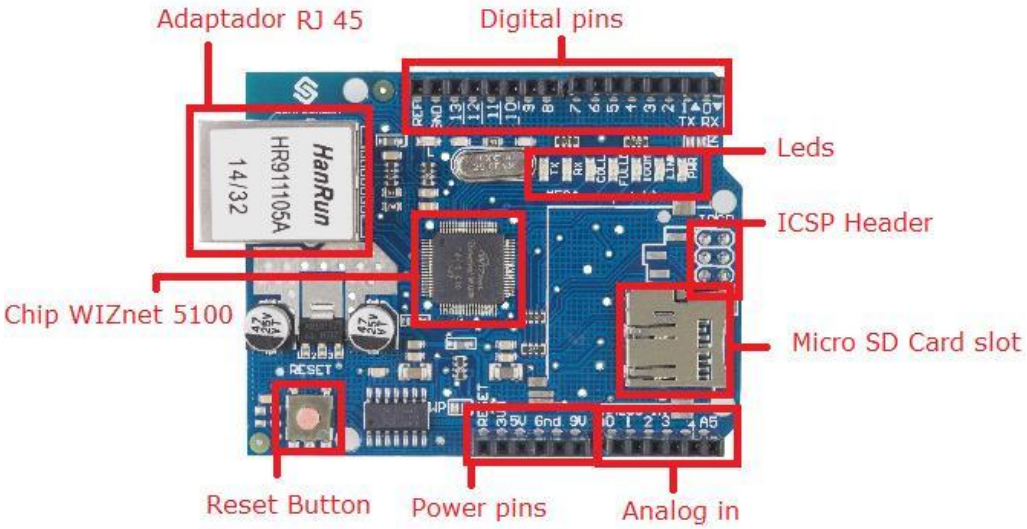

Figura 2.4: Ethernet Shield para arduino

Tabela 2.2: Leds indicadores do Shield Ethernet

PWR	indicates that the board and shield are powered
LINK	indicates the presence of a network link and flashes when the shield transmits or receives data
FULLHD	indicates that the network connections is full duplex
100M	indicates the presence of a 100 Mb/s network connection (as opposed to 10 Mb/s)
RX	flashes when the shield receives data
TX	flashes when the shield sends data
COL	flashes when network collisions are detected

IDE Arduino

A figura 2.5 mostra o IDE do Arduino, o Ambiente de desenvolvimento integrado que é utilizado para criar os rotinas que serão utilizadas pelo micro controlador ATmega328P para comandar sua aplicação.

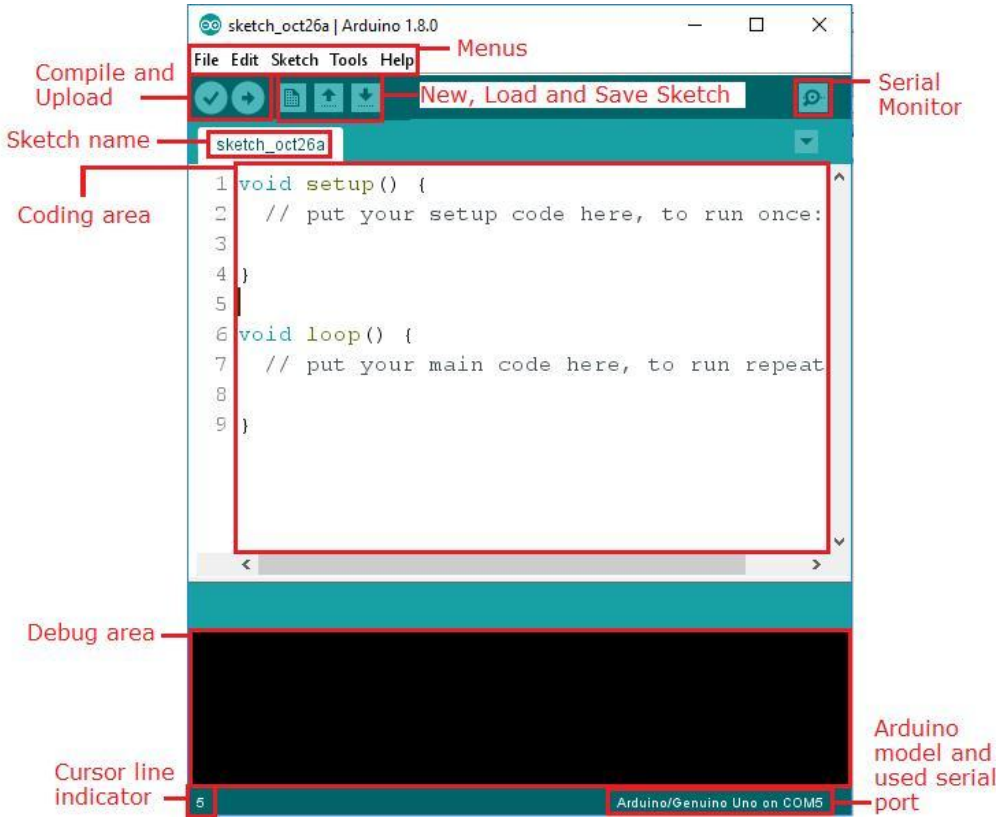

Figura 2.5: IDE do Arduino

Os códigos das rotinas no IDE do Arduino são escritos em C++, e algumas das bibliotecas de suporte estão em C/C++. Os programas criados são denominados sketches, e seu nome aparece ao topo da página. Logo acima, há uma aba com seis botões. O primeiro botão à esquerda serve para compilar a sketch, seguido pelo botão de upload, que carrega a sketch para a placa Arduino através de um cabo USB. Em seguida vem os botões de criar nova sketch, carregar uma sketch salva, e salvar sketch, nessa ordem. O último botão, no lado direito desta aba é o botão de monitoramento serial, que serve para monitorar o que o Arduino conectado à porta USB do envia para a saída serial, através do método "Serial.print()" ou "Serial.write()", possibilitando a análise em tempo real da aplicação. Abaixo desta aba há a área reservada para escrever o código, com uma coluna a esquerda indicando as linhas do código escrito. Abaixo da área de código fica a área de depuração, onde o IDE informa os erros de compilação ou carregamento para a placa, caso algum ocorra, e também informa quanto de memória flash e ram o programa utiliza do total disponível. Logo a esquerda no rodapé, temos a indicação da linha de código em que está o cursor do teclado, e do lado direito do rodapé temos a indicação do modelo da placa conectada a porta serial, e em qual porta ela está conectada.

Como pode-se notar na área de codificação da figura 2.5, a sketch possui duas funções em branco: função setup e função loop. Toda vez que uma nova sketch é criada, ela já vem com essas duas funções escritas, pois toda e qualquer sketch para Arduino deve conter essas duas funções, ambas do tipo "void", que significa que elas não retornam nenhum valor quando chamadas. A função setup é chamada automaticamente pelo ATmega328P toda vez que o programa for inicializado, seja pelo carregamento deste para a placa, ou quando reinicializado por um reset. Nela são realizadas todas as configurações iniciais do programa, como configuração das portas digitiais e analógicas, inicialização da comunicação serial, quando utilizada, e quaisquer outra configuração que seja necessária realizar assim que o programa iniciar. Assim que a função setup termina, a função loop será chamada continuamente até o fim da execução do programa, seja por uma parada forçada pelo próprio código, ou pelo desligamento da placa Arduino, caso contrário, a função loop continuará sendo chamada ininterruptamente toda vez que ela terminar de ser executada. Na função loop encontra-se o código responsável pelo comando da aplicação em si. É possível criar outras funções além das duas obrigatórias, e elas podem ser chamadas de dentro das funções setup e loop.

2.2 XBee Pro s2

O módulo XBee Pro s2 é um micro controlador produzido pela Digi International, e está representado na figura 2.6. A comunicação entre os módulos XBee segue o protocolo ZigBee (IEEE 802.15.4). A escolha desse protocolo foi feita baseado em seu alcance e sua resiliencia a interferência. Realizando um comparativo com os protocolos bluetooth e WiFi (IEEE 802.11), constata-se que ela possui alcance efetivo muito superior ao bluetooth, que possui apenas 10 metros contra 90 metros do protocolo ZigBee, e equivalente ao wifi , diferenciando-se deste pela melhor resiliencia à interferencia. O custo dos módulos porém é um pouco mais elevado, cerca de 100 reais por módulo XBee no varejo, contra 20 reais dos módulos bluetooth e 15 reais dos módulos WiFi ESP 8266. Apesar do maior custo, a confiabilidade dos módulos XBee em relação a interferência e seu maior alcance fornecem uma boa solução para a aplicação proposta. As comunicações por este protocolo são realizadas nas faixas de frequência que não necessitam de licença (ISM – industrial, scientific and medical radio band) com taxa de transferência de dados diferente para cada uma delas. O modulo XBee utilizado opera na faixa de 2.4 GHz, com 16 canais e taxa de transferência de 250 kbps.

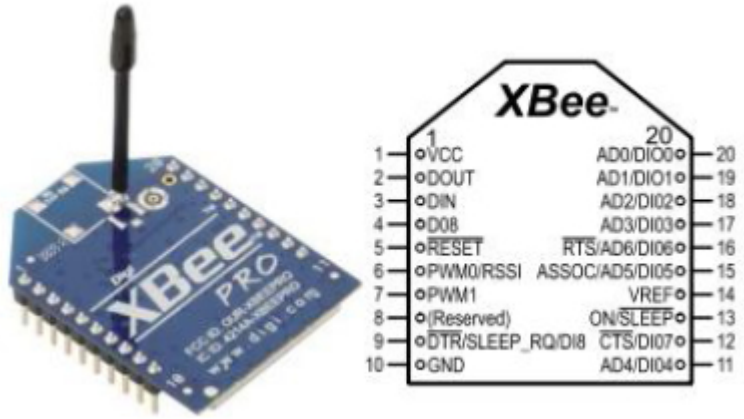

Figura 2.6: Módulo XBee Pro s2

A tabela 2.3 mostra a pinagem do módulo XBee Pro S2. Ele é composto por vinte pinos, sendo dois de alimentação: VCC (pino 1) e terra (pino 10), dois pinos de transferência de dados serial (pino 2 de saída e pino 3 de entrada) e doze pinos que podem ser configurados como entrada ou saída digitais (pinos 4, 6, 7, 9, 11, 12, 13, 15, 16, 17, 18, 19 e 20). Desses doze pinos, quatro, deles operam também como entrada analógica (pinos 17, 18, 19 e 20) e dois podem ser configurados pra realizar outras funções: o pino 13, de saída, que indica se o módulo está em repouso ou não, e o pino 9, de entrada, para controlar o modo de repouso do módulo. Além disso, há também outros dois pinos de entrada: um para reiniciar o módulo (pino 5) e um que fornece a tensão de referência para as entradas analógicas e digitais (pino 14).

Tabela 2.3: Pinagem do XBee Pro s2

Pin #	Name	Direction	Default State	Description
1	VCC	-	-	Power supply
2	DOUT	Output	Output	UART Data Out
3	DIN / CONFIG*	Input	Input	UART Data In
4	DIO12	Both	Disabled	Digital IO 12
5	RESET	Both	Open Collector with pull-up	Module Reset (reset pulse must be at least 200 ns)
6	RSSI PWM / DIO10	Both	Output	RX Signal Strnght Indicator or Digital IO 10
7	DIO11	Both	Input	Digital IO 11
8	[reserved]	-	Disabled	Do not connect
9	DTR* / SLEEP_RQ/ DIO8	Both	Input	Pin Sleep Control Line or Digital IO 8
10	GND	-	-	Ground
11	DIO4	Both	Disabled	Digital IO 4
12	CTS* / DIO7	Both	Output	Clear-to-Send Flow Control or Digital IO 7.
13	ON / SLEEP*	Output	Output	Module Status Indicator or Digital IO 9
14	VREF	Input	-	Reference Voltage pin connected when sampling an analog input
15	Associate / DIO5	Both	Output	Associated Indicator or Digital IO 5
16	RTS* / DIO6	Both	Input	Request-to-Send Flow Control or Digital IO 6.
17	AD3 / DIO3	Both	Disabled	Analog Input 3 or Digital IO 3
18	AD2 / DIO2	Both	Disabled	Analog Input 2 or Digital IO 2
19	AD1 / DIO1	Both	Disabled	Analog Input 1 or Digital IO 1
20	AD0 / DIO0 / Comissioning Button	Both	Disabled	Analog Input 0 or Digital IO 0

A tabela 2.4 mostra as especificações técnicas do módulo XBee PRO S2. Como se pode notar, ele possui alcance de 90 metros em ambientes fechados e áreas urbanas. Essa distância essa que aumenta para até 1500 metros em áreas abertas. Sua tensão de operação é de 3.3 volts, com potência de transmissão de 10 mW, e taxa de transferência de dados de 250 kbps através da comunicação por rádio frequência, e até 1 Mbps pela comunicação serial.

Essas características tornam este módulo uma excelente escolha em aplicações de automação utilizando redes sem fio. Principalmente pelo baixíssimo consumo, devido ao eficiente sistema de modo de repouso dos módulos, e também pelo alcance obtido com a sua utilização, muito superior ao bluetooth (10 metros) e superior ao WI-FI (70 metros). Para aplicações que utilizem áudios e vídeos, porém, essa não é a melhor alternativa, devido à baixa taxa de transferência de dados se comparado às outras tecnologias.

Tabela 2.4: Especificações técnicas XBee Pro s2

Specification	XBee Pro (s2)
Supply voltage	3.0 - 3.4 V
Operating Current (transmit)	295 mA (@3.3 V)
Operating Current (receive)	45 mA (@3.3 V)
Idle Current (receive off)	15 mA
Operating frequency band	ISM 2.4 Ghz
Dimensions	2.438 cm x 3.294 cm
Operating temperature	-40 to 85 Celsius (industrial)
Transmit power output	17 mW (+17 dBm) 10 mW (+10 dBm) for international variant
RF data rate	250.000 b/s
Data throughput	up to 35.000 b/s
Serial interface data rate	1200 b/s - 1 Mb/s (non-standard baud rates also supported)
Indoor / urban range	Up to 300 ft. (90 m) Up to 200 ft. (60 m) international variant
Outdoor RF line of sight range	Up to 2 miles (3200 m) Up to 5000 ft (1500 m) international variant

Através da integração dos módulos XBee, é possível criar redes sem fio de diversas topologias, como: par, estrela, malha e árvore. Cada módulo atua como um nó da rede, e pode realizar três possíveis papéis: coordenador, roteador ou dispositivo final. Esses papéis servem para definir uma hierarquia de comunicação entre os módulos, e também para definir as funções que cada um deles deverá exercer na mesma. A figura 2.7 abaixo ilustra um exemplo de rede Zigbee e os papéis exercidos pelos módulos na rede: coordenador (C), roteador (R) e dispositivo final (E).

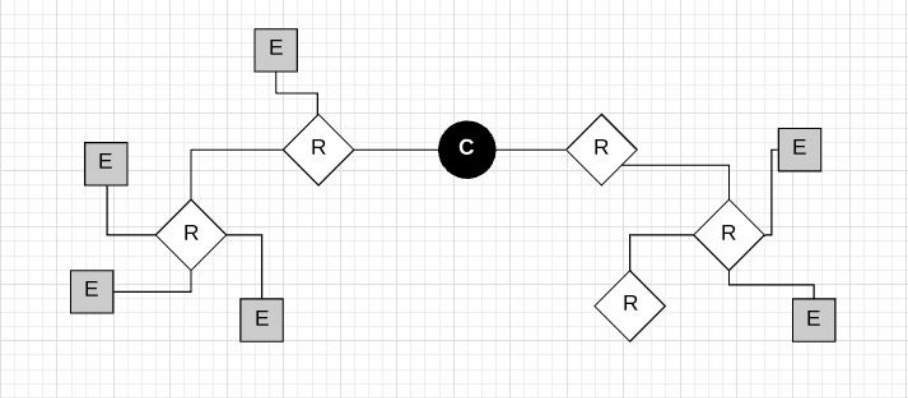

Figura 2.7: Redes Zigbee

Cada rede ZigBee deverá possuir um único coordenador, que é responsável por configurar o resto da rede. Ele é responsável por configurar as características únicas da rede local ZigBee, permitir entrada de roteadores e dispositivos finais na rede, e auxilia no roteamento de dados, nunca podendo entrar em modo de economia de energia.

Os roteadores podem ser vários em uma rede ZigBee e eles realizam basicamente as mesmas funções do coordenador, mas, diferentemente do coordenador, ele não participa das configurações da rede. Após se juntar a rede, ele pode permitir a entrada de outros roteadores e dispositivos finais ligados a ele e auxilia no roteamento de dados.

Os dispositivos finais também podem ser inúmeros numa rede ZigBee e geralmente são utilizados juntos a sensores e atuadores devido à sua capacidade de entrar em modo de economia de energia. Ao contrário do coordenador e dos roteadores, porém, eles não podem auxiliar no roteamento de dados.

Em uma rede ZigBee os módulos XBee possuem ainda dois modos de operação: AT e API. O modo de operação AT é o mais simples, e é utilizado para módulos desempenhando papel de dispositivo final, ou em redes com topologia par. Um módulo configurado desta maneira pode-se conectar a apenas um outro módulo, e todo dado recebido pelo seu pino de entrada serial, ou pelos dados recebidos nas portas de entrada solicitados por outro módulo, são transmitidos ao módulo adjacente via radio frequência, como mostrado na figura 2.8.

Figura 2.8: Modo de operação AT

O modo de operação API é mais complexo, pois permite ao XBee se conectar a vários XBees adjacentes, sejam eles API ou AT. Neste modo de operação, os módulos podem enviar frames de bytes para os outros módulos da rede, permitindo assim o envio de comandos ou dados a diversos módulos da rede, e não apenas a um módulo como acontece no modo AT. Esses frames possuem a estrutura padrão mostrada na figura 2.9, sendo composto por um byte inicial, o tamanho do frame, e os bytes correspondentes a cada tipo de frame disponível. A tabela 2.5 lista os tipos de frames API existentes.

Figura 2.9: Estrutura dos frames API

Dentre os tipos de frame disponíveis, apenas dois foram utilizados neste trabalho: Remote AT Command e Zigbee Transmit Request. O Remote AT command é o tipo de frame utilizado quando se deseja realizar uma configuração remota de um outro módulo, como por exemplo acionamento das portas digitais ou alteração de configurações padrão. A tabela 2.7 mostra um exemplo de remote AT command, em que a porta digital D1 é configurada como em estado alto (3.3 volts). O Zigbee Transmit Request é o tipo de frame utilizado quando se deseja enviar bytes de dados a um outro módulo da rede. A tabela 3.3 mostra um exemplo em que é enviado oito caracteres ("fanpower") de dados para o módulo XBee conectado ao módulo capaz de controlar o ventilador de teto.

Tabela 2.5: Tipos de frame API

API Frame Names	API ID
AT Command	0x08
AT Command - Queue Parameter Value	0x09
Zigbee Transmit Request	0x10
Explicit Addressing Zigbbe Command Frame	0x11
Remote Command Request	0x17
Create Source Route	0x21
AT Command Response	0x88
Modem Status	0x8A
Zigbee Transmit Status	0x8B
Zigbee Receive Packet (AO=0)	0x90
Zigbee Explicit Rx Indicator (AO=1)	0x91
Zigbee IO Data Sample Rx Indicator	0x92
XBee Sensor read Indicator (AO=0)	0x94
Node Identification Indicator (AO=0)	0x95
Remote Command Response	0x97
Over-the-air firmware update status	0xA0
Route Record Indicator	0xA1
Many-to-one Route Request Indicator	0xA3

2.2.1 Configuração dos XBees

Para configurar uma rede ZigBee, é preciso configurar cada um dos módulos presente na mesma individualmente. Essa configuração pode ser realizada de três maneiras distintas: através do software Digi XCTU, através de comandos enviados pela porta serial, ou através de comandos por radio frequência a partir de um módulo operando em modo API, adjacente ao módulo a ser configurado, ou pelo coordenador da rede.

Configuração pela porta serial

A configuração via porta serial pode ser utilizada tanto para alterar parâmetros de módulos já conectados à rede, como para configurar módulos que ainda não estão conectados à ela. Para configurar ou ler os parâmetros do módulo, é necessário que o mesmo entre em modo de comando, modo no qual os caracteres na porta serial são interpretados como comandos. Para entrar no modo de comando, deve-se enviar a sequencia de caracteres "+++" na porta de entrada serial, e aguardar a resposta "ok\r" na saída serial. Dentro do modo de comando, basta utilizar a sequência ilustrada na figura 2.10, composta pelo prefixo "AT", a sigla do comando composta por dois caracteres, um espaço, o parâmetro em hexadecimal (caso esteja em branco configura uma leitura daquele parâmetro), seguido por ultimo dos caracteres de

Tabela 2.6: Exemplo de Zigbee Transmit Request

Frame Fields		Byte	Example	Description
Start Delimiter		0	0x7E	Byte inicial
Lenght		1	0x00	Número de bytes do frame-specific-data
		2	0x16	
Frame-specific Data	Frame Type	3	0x10	Tipo de frame (Zigbee transmit request)
	Frame ID	4	0x01	Mensagem de status de transmissão 0x00 = desabilita mensagem
	64-bit Destination Address	5	0x00	Endereço do módulo XBee de destino do Zigbee Transmit Request
		6	0x13	
		7	0xA2	
		8	0x00	
		9	0x40	
		10	0x9B	
		11	0x12	
		12	0x6B	
	16-bit Destination Network Address	13	0xFF .	Endereço da rede de destino do Zigbee transmit request FFFE = broadcast
		14	0xFE .	
	Broadcast Radius	15	0x00	Define número máximo de saltos de uma transmissão broadcast. 0x00 = número máximo de saltos
	Options	16	0x00	Opções de transmissão 0x00 = opções padrão
	RF Data	17	f	Dados a serem transmitidos
		18	a	
		19	n	
		20	p	
		21	o	
		22	w	
		23	e	
		24	r	
Checksum		25	checksum	0xFF - 8 bits da soma dos bytes do Frame-specific-data

Tabela 2.7: Exemplo de comando AT remoto

Frame Fields		Byte	Example	Description
Start delimiter		0	0x7E	Byte inicial
Lenght	MSB	1	0x00	Comprimento da mensagem
	LSB	2	0x10	(Número de bytes do frame specific data)
Frame-Specific data	Frame Type	3	0x17	Especifica o tipo de frame
	Frame ID	4	0x00	Mensagem de status de transmissão 0x00 = desabilita mensagem
	64-bit Destination Address	5	0x00	Endereço do módulo XBee de destino do comando AT
		6	0x13	
		7	0xA2	
		8	0x00	
		9	0x40	
		10	0x9B	
		11	0x12	
		12	0x5F	
	16-bit Destination Network Address	13	0xFF	Endereço da rede de destino do comando FFFE = broadcast
		14	0xFE	
	Remote Command Options	15	0x02	Opções de comando remoto 0x02 = aplicar mudanças
	AT Command	16	0x44	Comando AT D1 (ASCII) = 4431 (hexadecimal)
		17	0x31	
	Command Parameter	18	0x05	Parametro do comando enviado 0x05 = Saída digital alta (3.3 V)
Checksum	Checksum	19	0x6E	Byte de verificação

conclusão do comando "\r". Por exemplo, o comando representado na figura 2.10, "ATD1 05\r", possui os dois primeiros dígitos referentes ao comando AT, depois os caracteres "D1", referentes à configuração da porta de entrada e saída digital 1, seguido pelos dois caracteres de parâmetro "05", equivalentes a configurar a porta como saída em estado alto (3.3v), e por último os caracteres "\r" de retorno. Para cada comando enviado, uma mensagem de confirmação será emitida em resposta, assim como ao entrar no modo de comando. Para que os comandos fiquem salvos na memória do módulo após um reset físico futuro, deve-se enviar o comando "WR" (write), logo após a resposta de confirmação. No apêndice há a tabela que mostra todos os comandos AT existentes e suas siglas e parâmetros.

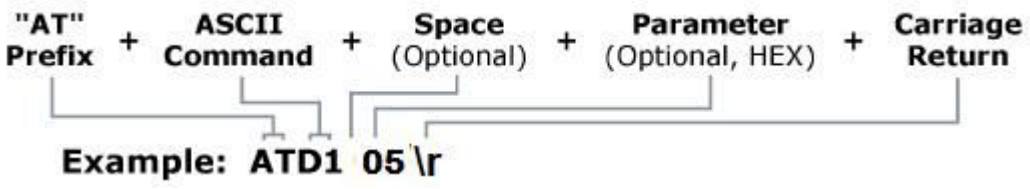

Figura 2.10: Exemplo de comando AT

Configuração pelo Digi XCTU

O software Digi XCTU, oferece uma interface mais amigável e fácil para a configuração dos módulos XBee, e assim como na configuração via serial, pode-se tanto alterar parâmetros de módulos já conectados, como também configurar

módulos que ainda irão se juntar à rede. O software irá converter os parâmetros definidos nele em comandos que serão enviados pela porta serial, como mostrado na configuração pela porta serial. No exemplo ilustrado na figura 2.11, está sendo realizada a configuração da porta digital 1 como saída e definindo seu estado lógico como alto (3.3v). Para configurar o módulo através do software, é necessária a utilização do XBee Explorer, representado na figura 2.12, um adaptador serial que conecta o módulo XBee ao computador via USB.

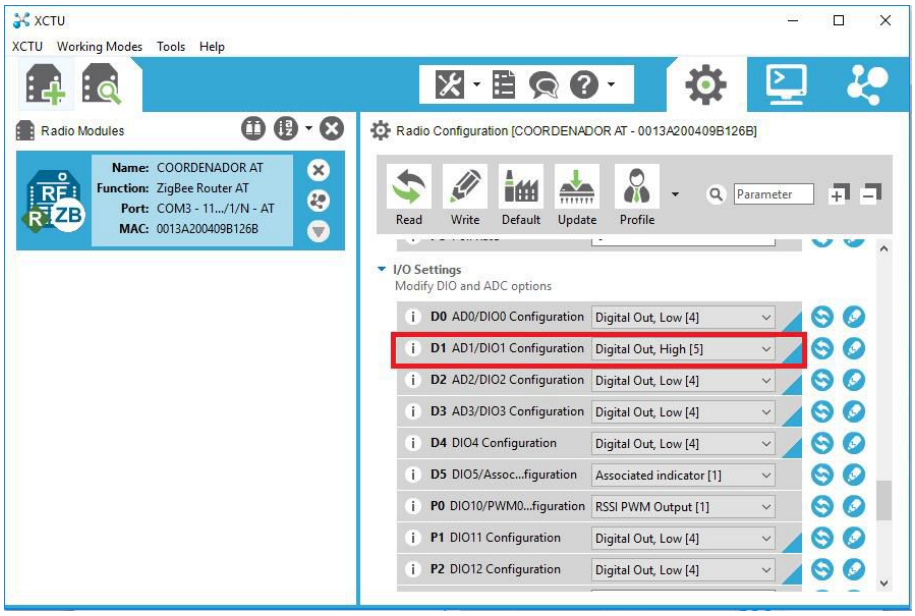

Figura 2.11: Digi XCTU

Figura 2.12: XBee Explorer

Configuração remota

Ao contrário da configuração via porta serial e através do software, a configuração remota é utilizada somente para alterar parâmetros de módulos que já estão conectados à rede ZigBee. Ela é realizada a partir de um módulo operando em modo API, que envia os comandos AT via radio frequência para o módulo que se deseja configurar. O comando AT é enviado dentro de um frame com vinte bytes, ilustrado na tabela, que especifica os campos do frame nas duas primeiras colunas, a numeração dos bytes na terceira, os valores de exemplo para configurar a porta digital D1 como porta de saída em estado alto na quarta coluna, e por ultimo a descrição dos bytes. O frame é composto por um byte de inicio da mensagem, dois bytes para determinar o tamanho em bytes da mensagem, um byte para especificar se a mensagem é um comando ou uma requisição de dados, um byte para requisitar status de envio da mensagem, oito bytes do endereço de destino, dois bytes do endereço da rede de destino, um byte de opções de comandos remotos, dois bytes para o comando AT (um para cada caractere da sigla do comando), um byte com o valor do parâmetro a ser mudado, e um byte final de verificação do numero de bytes da mensagem. Este último byte deve ser calculado corretamente, caso contrário a mensagem não é enviada. Para calcular o byte checksum, deve-se somar todos os bytes de 3 a 18, selecionar apenas os 8 bits menos significativos e subtrair de 0xFF. No exemplo da tabela 2.7, a soma dos bytes em hexadecimal é 0492, cujos 8 bits menos significativos são 1001 0010. Subtraindo isso de 1111 1111, temos: 0x6E em hexadecimal.

2.3 MIT App Inventor 2

O MIT App Inventor é um ambiente de programação visual de aplicativos para smartphones e tablets que utilizam sistema operacional Android, e foi desenvolvido com o objetivo de tornar a programação de aplicativos acessível a todos, incluindo pessoas que não têm conhecimento de programação e até crianças. Devido à sua ferramenta baseada em blocos, que são arrastados com o mouse do computador para adicionar as funcionalidades do aplicativo e para criar a lógica que comanda o programa, é possível criar aplicativos sem ser preciso escrever uma única linha de comando. Sua escolha foi realizada visando otimizar o tempo disponível para a realização do projeto, devido a sua simplicidade de desenvolvimento. Como o foco do projeto era o desenvolvimento do sistema de automação como um todo e não apenas o aplicativo, a utilização desta ferramenta permitiu o desenvolvimento de um aplicativo funcional em bem menos tempo.

O ambiente permite ainda o acompanhamento e teste em tempo real do aplicativo que está sendo criado. Para isso, é necessária a utilização de um emulador de android no computador (aiStarter), ou a utilização do aplicativo MIT AI2 Companion, que pode ser baixado para seu smartphone ou tablet. Para realizar o acompanhamento em tempo real, basta clicar no menu "Connection" e selecionar a opção desejada: "AI Companion", para fazer upload dos arquivos referentes ao aplicativo para um smartphone ou tablete que esteja conectado à mesma rede WIFI, "USB", para enviar os arquivos pela porta USB para o dispositivo desejado, ou "Emulator", para transferir os dados do browser para o computador. O aplicativo irá aparecer na tela do dispositivo ou emulador, e qualquer mudança efetuada no browser será realizada em tempo real no dispositivo.

O ambiente de desenvolvimento do MIT App Inventor é dividido em duas seções: "design" e "blocks". Na seção design, representada na figura 2.13, o usuário adiciona e configura todos os componentes e funcionalidades desejadas em seu aplicativo, organiza a interface gráfica do mesmo e adiciona mídias que serão utilizados na sua execução. Ela é dividida em quatro áreas: Palette, Viewer, Components e Properties. Na área Palette (retângulo amarelo) encontram-se todos os componentes disponíveis para desenvolvimento do aplicativo, e é dividida em onze tipos de componentes: user interface, layout, media, drawing and animation, sensors, social, storage, connectivity, lego mindstorms, experimental e extension. Para adicionar qualquer componente ao seu aplicativo, basta clicar sobre ele com o mouse e arrastá-lo para a área Viewer (retângulo vermelho), área em que é possível organizar a disposição em que os componentes aparecerão no aplicativo. Quando o componente é adicionado à área Viewer, ele automaticamente aparece listado na área Components (retângulo verde), que serve para selecionar os diferentes componentes que fazem parte do aplicativo. Por útltimo, a área Properties serve para configurar as propriedades dos componentes selecionados na área Components. Cada componente possui características distintas, e portanto, diferentes propriedades que podem ser configuradas.

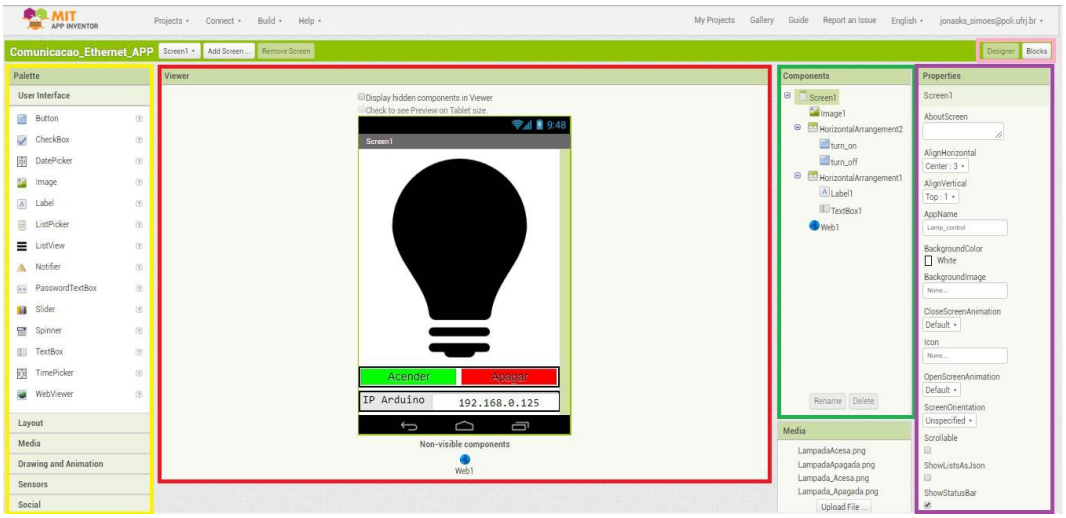

Figura 2.13: Seção Designer do MIT App Inventor 2

Já na seção "blocks", mostrada na figura 2.14, o usuário estrutura a lógica e o funcionamento do aplicativo através da integração dos variados componentes que foram adicionados na seção de design. Essa estruturação é realizada através de blocos de texto, lógica, variáveis, cores, dentre outros, que são organizados pelo usuário utilizando o mouse do computador para arrastá-los para a área de criação, conectando um bloco ao outro. Esta seção é dividida em duas áreas: Viewer (retângulo vermelho) e Blocks (retângulo amarelo). Na área blocks estão contidos todos os blocos que serão utilizados para determinar o funcionamento do aplicativo. Nela estão disponíveis blocos built-in, que são responsáveis pela lógica funcional do aplicativo, e blocos referentes aos componentes adicionados na seção de design. Dentre os blocos built-in, existem: blocos de controle, lógica, matemática, texto, listas, cores, variáveis e procedimentos. Ao clicar sobre qualquer das opções de blocos disponíveis, uma paleta irá se abrir (retângulo verde), mostrando todas as opções de blocos referentes àquela opção. A figura 2.15, ilustra o arranjo ampliado presente da área viewer da figura 2.14, que determina o comportamento do aplicativo quando o botão "turn_off" do aplicativo for pressionado. Nesse arranjo, tem-se um bloco do componente Botão, dois blocos do componente WEB e três blocos de texto. Ao clicar no botão "turn_off", botão vermelho na área Viewer da figura 2.13, o bloco marrom é ativado, completando assim a ação determinada nos blocos englobados por ele. Primeiro, temos o bloco verde, que configura a URL do componente WEB para o texto contido nos blocos em cor rosa (união dos blocos de texto à direita do bloco verde), formando a URL http://192.168.0.200/80/ReqL01_0fim. Em seguida há a ativação do bloco roxo que envia uma solicitação GET para a URL citada, que corresponde à URL do web server rodando no arduino, como será visto a seguir.

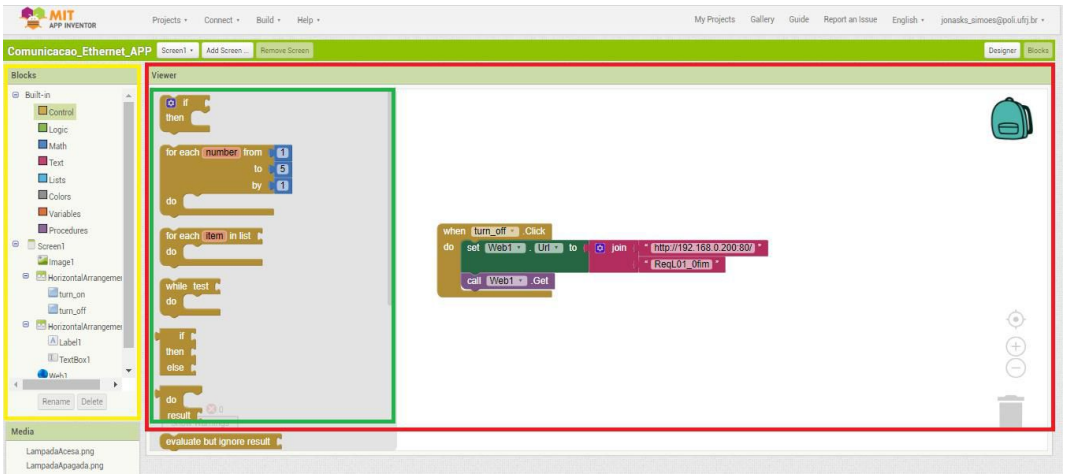

Figura 2.14: Seção Blocks do MIT App Inventor 2

Figura 2.15: Blocos e lógica de programação do MIT App Inventor 2

Capítulo 3

Implementação do sistema de automação residencial

Neste capítulo será abordada a metodologia utilizada para a implementação do sistema de automação residencial utilizando a plataforma Arduino e o protocolo de comunicação por rádio frequência Zigbee. Inicialmente será explicado a arquitetura do sistema, que é composto por um módulo central, um módulo de atuador relé e um modúlo de atuador infravermelho. Serão apresentados também os componentes físicos e os códigos de programação utilizados em cada um desses módulos. Em seguida, será mostrado como foi configurada a rede Zigbee para realizar a comunicação entre os módulos do sistema. Por fim, será explicado o funcionamento do aplicativo para smarthphone e a lógica de programação utilizada em seu desenvolvimento.

3.1 Arquitetura

O método proposto o sistema de automação residencial consiste na substituição do acionamento de aparelhos elétricos e eletrônicos, e iluminação de um ambiente através de aplicativo em celulares ou tablets. O sistema, representado na figura 3.1, possui uma arquitetura centralizada, em que os comandos são recebidos pelo módulo central, que interpreta, e os redireciona aos módulos atuadores, que são encarregados de controlar os dispositivos finais (lâmpadas, ar condicionado e ventilador de teto). São dois os tipos de módulos atuadores: módulo atuador de relés, responsável pelo acionamento das lâmpadas e o módulo atuador por infravermelho, responsável por enviar os sinais infravermelhos ao ar condicionado e ao ventilador de teto.

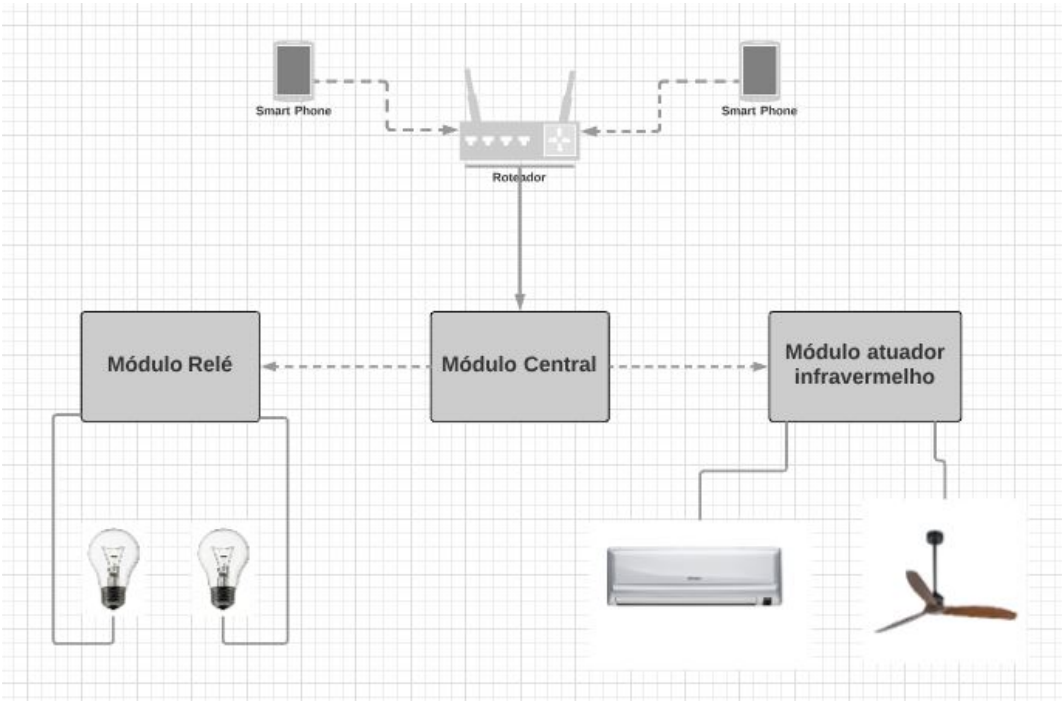

Figura 3.1: Arquitetura do sistema de automação residencial

A comunicação entre os usuários, através do aplicativo, e o sistema se dá através de um servidor web, criado no módulo central. Ele possibilita o envio de comandos do aplicativo ao módulo central do sitema através da internet, ou através da rede local mesmo que sem acesso à internet. A comunicação entre os módulos do sistema, por sua vez, é realizada através de rádio frequência, via protocolo zigbee como apresentado na seção 2.2, através dos XBees presentes em cada módulo do sistema, que formam uma rede zigbee.

3.2 Rede Zigbee

A rede Zigbee utilizada no projeto possui uma topologia bem simples, visto que é composta apenas por 3 XBees, sendo: um XBee conectado ao módulo central do sistema de automação na função de coordenador da rede em modo API, e dois XBees conectados aos módulos atuadores na função de roteador, sendo um em modo AT e outro em modo API. A configuração dos XBees foi feita utilizado o software digi XCTU e um XBee explorer. Cada XBee foi configurado individualmente, mas todos eles utilizando os mesmos procedimentos, que serão mostrados a seguir. O primeiro passo consiste em adicionar o XBee ao programa. Para isso deve-se clicar no ícone envolvido em vermelho na figura 3.2, e selecionar a porta USB onde o XBee está conectado, como na figura 3.3, apertando “next” em seguida.

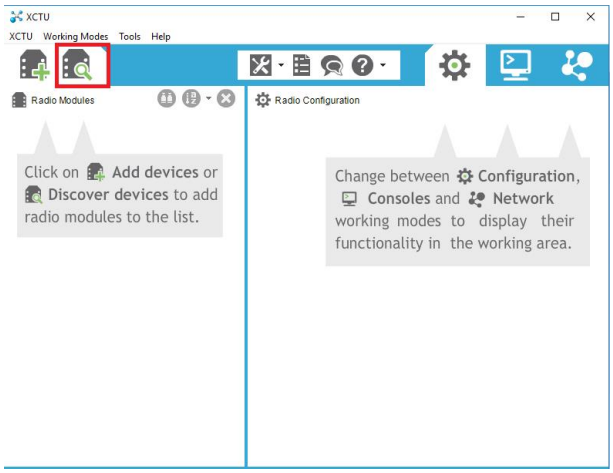

Figura 3.2: Janela do Digi XCTU

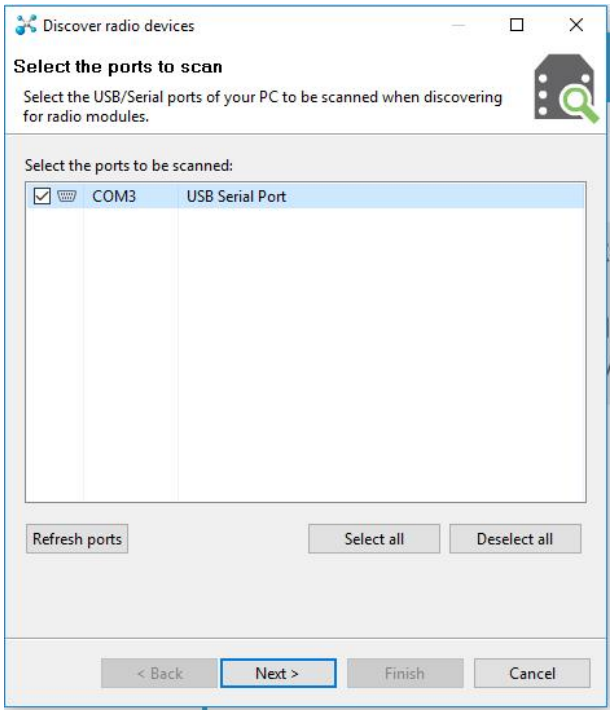

Figura 3.3: Janela de seleção da porta serial

A janela, mostrada na figura 3.4, se abre para definição dos parâmetros do XBee que se deseja localizar. Caso o módulo XBee ainda esteja com as configurações de fábrica, como no caso deste projeto, selecione “set defaults”. Caso não se saiba a configuração daquele XBee, escolha “select all”, ou caso saiba as configurações do XBee, selecione as caixinhas correspondentes e clique em “finish”. Note que ao escolher "select all", o tempo de busca irá aumentar consideravelmente.

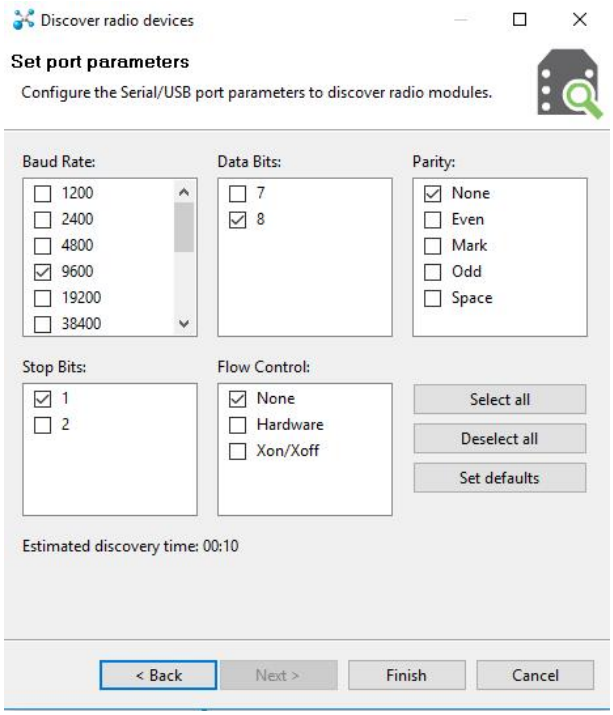

Figura 3.4: Janela de seleção dos parâmetros do xbee procurado

Caso aconteça que algum XBee esteja com o firmware corrompido ou em modo de repouso, a janela mostrada na 3.5 será exibida e será necessário realizar um reset físico no dispositivo. Para tal, utilize o botão de reset do XBee Explorer caso o ele possua, ou, caso contrário, utilize um jumper para ligar o pino de reset do modulo XBee (pino 5) ao pino ground (pino 10). Note que ligar os pinos de reset e ground do XBee explorer não tem o mesmo efeito.

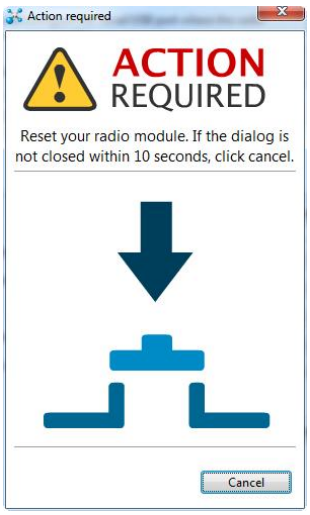

Figura 3.5: Janela de erro solicitando reset físico do xbee

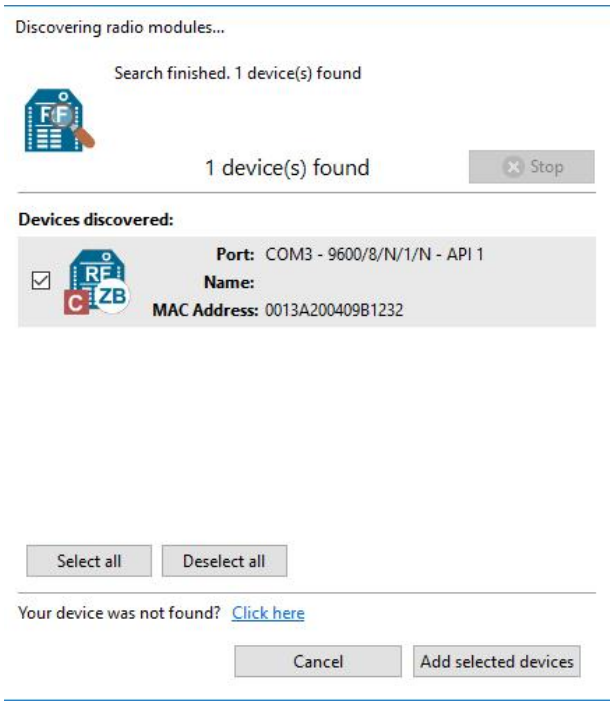

Figura 3.6: Janela com dispositivos encontrados

Os dispositivos localizados são listados na janela ilustrada na figura 3.6. Para adicioná-lo, basta selecioná-lo e clicar em "add selected devices". O XBee adicionado será exibido dentro do retângulo vermelho ilustrado na figura, e após clicar sobre ele, suas informações serão lidas pelo programa e posteriormente exibidas no retângulo amarelo, onde todas as configurações desejadas podem ser realizadas. Para escolher a função e modo de operação do XBee, deve-se clicar no ícone "update", que irá abrir a janela mostrada na figura 3.7. Escolhe-se então o modelo do XBee, a função que deseja configurá-lo e o firmware a ser utilizado por ele.

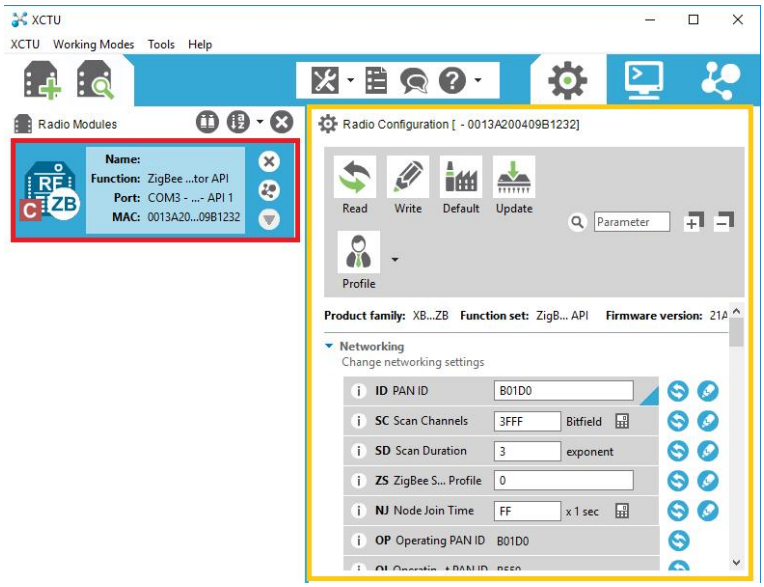

Figura 3.7: Configurações gerais dos XBees

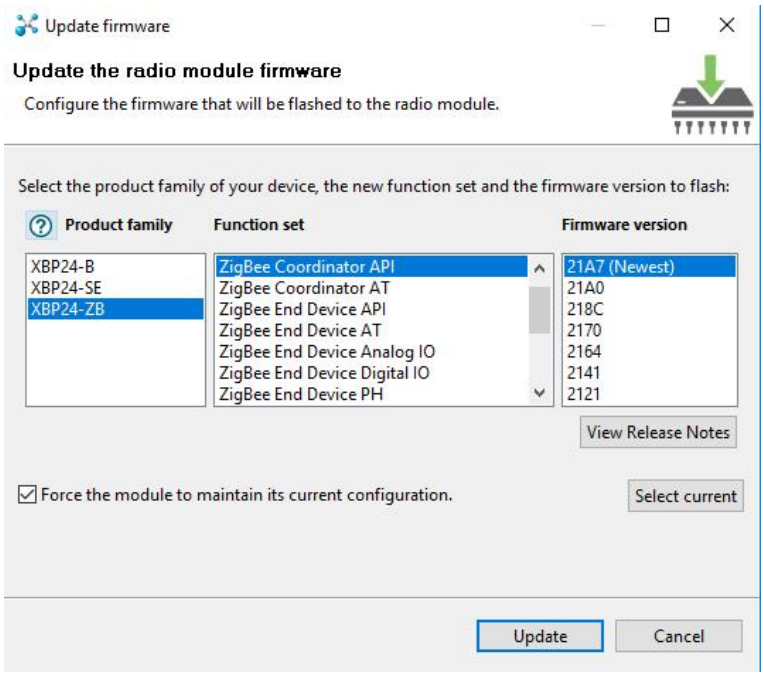

Figura 3.8: Configuração do modo de operação do XBee

As demais configurações podem ser todas realizadas na área do retângulo laranja citada anteriormente. Os módulos utilizados neste projeto foram todos configurados com mesmo PAN ID (AAA000), para que pertençam à mesma rede, e foram mantidas todas as configurações de fábrica, menos as mostradas na tabela 3.1 para cada XBee.O XBee conectado ao módulo central foi configurado como coordenador API, podendo assim enviar frames API para cada um dos módulos atuadores. O XBee conectado aos relés que controlam as lâmpadas foi configurado

Tabela 3.1: Configuração dos módulos XBee

	Modo de operação	Configurações alteradas
Módulo Central	Coordenador API	ID: AAA000
Módulo do atuador relé	Roteador AT	ID: AAA000 D0: Digital Output, LOW D1: Digital Output, LOW
Módulo do atuador infravermelho	Roteador API	ID: AAA000

como roteador em modo AT, para que possa receber frames de comandos AT remotamente. Além disso, ele teve todas as portas digitais desabilitadas, menos as portas D0 e D1, configuradas como pinos de saída digital, para controle dos dois relés. Por fim, o XBee conectado ao módulo infravermelho foi configurado como roteador API, para poder receber dados através de frames de transmit request, e teve todas os pinos digitais desabilitados.

3.3 Módulo central

O módulo central é composto por uma placa Arduino Uno com o ethernet shield acoplado e conectada a um XBee através de jumpers, como visto na figura 3.9. O shield ethernet possibilita a conexão do Arduino ao modem da residência através de um cabo ethernet, enquanto que o fica responsável por permitir a comunicação do módulo central com os módulos atuadores (relé e infravermelho) da residência por rádio frequência. O arduino por sua vez, fica encarregado de criar um servidor web, através do qual os comandos oriundos dos dispositivos dos usuários chega ao sistema de automação, interpretar esses comandos e redirecioná-los aos módulos necessários. Como visto na seção 2.2, a alimentação do XBee se dá pela conexão do pino 1 à saída 3.3V do arduino, e o pino 10 ao terra, enquanto que a comunicação serial se dá pela conexão dos pinos Tx e Rx do arduino aos pinos Din e Dout, respectivamente.

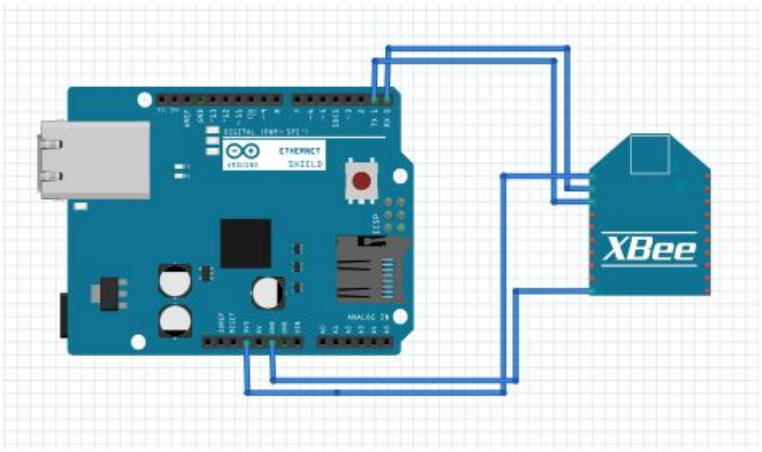

Figura 3.9: Esquema de montagem do módulo relé

A configuração do módulo central foi realizada em algumas etapas. Primeiramente, configurou-se o XBee, como visto na seção 3.2, para que ele seja coordenador em modo API, permitindo assim enviar frames com dados ou comandos para os demais XBees do sistema.

Em seguida criou-se a sketch para o arduino do módulo central, disponível no apêndice. A sketch "Modulo_central" é composta pelas duas funções obrigatórias setup e loop, além de quatro outras funções: redirect_commands, set_relay, set_ir e send_app.

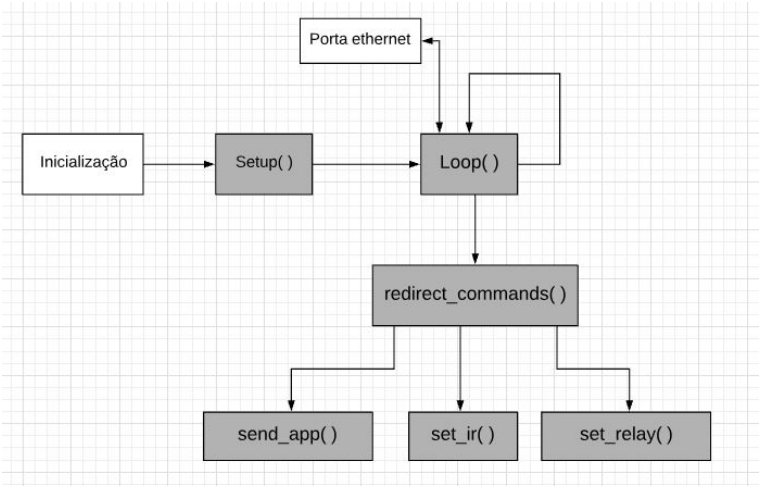

Figura 3.10: Fluxograma de funcionamento da sketch do módulo central

A etapa de inicialização é responsável pela inclusão de bibliotecas e declaração e/ou definição das variáveis globais, e é a primeira etapa que acontece ao ligar a placa arduino. Ao fim da inicialização o Arduino chama a função *Setup*, onde são inicializados a comunicação serial, responsável pela comunicação do arduino com o

XBee, com taxa de 9600 bits por segundo, e o servidor web, que deve possuir os 7 primeiros dígitos iguais ao da rede local a que se deseja conectar, e o número que sucede o último ponto final distintos dos outros dispositivos da rede (devendo este ser menor que 255). Além disso, o MAC address utilizado deve ser único na rede. Neste trabalho, utilizou-se o endereço IP 192.168.0.103, pois o IP do modem utilizado era 192.168.0.1.

A função *Loop* faz a leitura do servidor web à procura de dados. Caso encontre dados disponíveis, ela preenche o buffer (buf) com eles, extrai o comando enviado desses dados, e o envia para a função redirect_commands através de um array de oito caracteres. Em seguida esvazia os buffers e variáveis e reinicia o loop. Caso não haja dados, ela continua o loop normalmente até que algum dado esteja disponível.

A função *redirect_commands* recebe um array de oito caracteres com o comando enviado pela função loop, decodifica o comando, e o encaminha ao modulo atuador através das funções *set_ir* e *set_relay*. Em seguida ela envia a resposta ao aplicativo através da função *send_app* e se encerra, voltando assim à função *Loop*.

A função *set_relay* é responsável por criar um frame API para envio de comando AT remoto ao XBee localizado no módulo atuador relé. Ela recebe o comando com a lâmpada que se deseja controlar e a ação desejada (ligar ou desligar), e envia um frame API do tipo remote AT command (frame type 0x10), para controlar os relés correspondentes.O frame criado nesta função para ser enviado ao módulo de atuador relé está disponível na tabela 3.2. O primeiro byte é o byte inicial (0x7E), seguido por dois bytes que indicam o tamanho do frame entre o byte numero 3 e o byte de checksum (0x10 para 16 bytes). O byte seguinte indica o tipo de frame que está sendo enviado (0x17 para remote AT command). O quinto byte determina o tipo de resposta esperada (0x00 indica que não há resposta). Os bytes de 6 a 13 contém o endereço físico do XBee ao qual se envia o frame (O XBee no módulo infravermelho tem endereço 0013A200409B125F. Os bytes 14 e 15 contém o endereço da rede Zigbee, que no caso foi definido como envio broadcast (FFFE). Os bytes 16 e 17 definem o tipo de comando AT ('D' e '1' para escolher a configuração da porta digital D1). Em seguida vem os byte do parâmetro do comando escolhido nos bytes 16 e 17 (0x04 para porta digital em 0 volts e 0x05 para porta digital em 3.3 volts). Por fim, o byte de checksum para confirmação de fim do frame.

A função *set_ir*, por sua vez, cria um frame API para envio de dados ao XBee localizado no módulo atuador infravermelho. Ela recebe o comando de oito caracteres e reenvia este comando através de um frame API do tipo zigbee

Tabela 3.2: Remote AT command API Frame

Byte 1	0x7E
Byte 2	0x00
Byte 3	0x16
Byte 4	0x17
Byte 5	0x01
Byte 6	0x00
Byte 7	0x13
Byte 8	0xA2
Byte 9	0x00
Byte 10	0x40
Byte 11	0x9B
Byte 12	0x12
Byte 13	0x5F
Byte 14	0xFF
Byte 15	0xFE
Byte 16	0x00
Byte 17	'D'
Byte 18	'1'
Byte 19	0x05
Byte 20	0x6E

transmit request (Frame type 0x17), que será interpretado pelo módulo atuador infravermelho para emitir o sinal infravermelho adequado. O frame criado nesta função para ser enviado ao módulo de atuador infravermelho está disponível na tabela 3.3. O primeiro byte é o byte inicial (0x7E), seguido por dois bytes que indicam o tamanho do frame entre o byte numero 3 e o byte de checksum (0x16 para 22 bytes). O byte seguinte indica o tipo de frame que está sendo enviado (0x10 para zigbee transmit request). O quinto byte determina o tipo de resposta esperada (0x00 indica que não há resposta). Os bytes de 6 a 13 contém o endereço físico do XBee ao qual se envia o frame (O XBee no módulo infravermelho tem endereço 0013A200409B126B. Os bytes 14 e 15 contem o endereço da rede Zigbee, que no caso foi definido como envio broadcast (FFFE). Os bytes 16 e 17 definem o numero de pontos da rede para trasmissão (0x00 para máximo de pontos) e as opções do envio (0x00 para nenhuma opção especial). Em seguida vem os 8 bytes de dados que possuem o comando a ser enviado, que no exemplo da tabela é um comando de oito caracteres para ligar ou desligar o ventilador(fanpower). Por último, o byte de checksum para confirmação de fim do frame. A função *send_app*, por sua vez, é responsável por criar um código HTML com a resposta a ser enviada ao aplicativo, com a confirmação de reenvio daquele comando. A resposta, quando recebida pelo aplicativo será interpretada para atualizar o status do sistema no mesmo.

Tabela 3.3: Zigbee transmit request API Frame

Byte 1	0x7E
Byte 2	0x00
Byte 3	0x16
Byte 4	0x10
Byte 5	0x01
Byte 6	0x00
Byte 7	0x13
Byte 8	0xA2
Byte 9	0x00
Byte 10	0x40
Byte 11	0x9B
Byte 12	0x12
Byte 13	0x6B
Byte 14	0xFF
Byte 15	0xFE
Byte 16	0x00
Byte 17	0x00
Byte 18	'f '
Byte 19	'a'
Byte 20	'n'
Byte 21	'p'
Byte 22	'o'
Byte 23	'w'
Byte 24	'e'
Byte 25	'r'
Byte 26	0x82

3.4 Módulos atuadores

O sistema de automação residencial implementado conta com dois módulos distintos de atuadores: o módulo de atuador relé e o módulo de atuador infravermelho. Ambos os módulos possuem XBees para se comunicar com a central, mas sem comunicação entre eles. Os módulos recebem os comandos redirecionados pela central e atuam de maneira a controlar os dispositivos finais.

3.4.1 Módulo de atuador relé

Os módulos relé atuam no controle das lâmpadas e foram implementados para substituir o acionamento mecânico através dos interruptores. Como mostra a figura 3.11, este módulo é composto por um XBee e um relé de dois canais (um para cada lâmpada), um regulador de tensão (para alimentar o XBee com 3.3 volts) e é conectado a duas lâmpadas, representadas na figura por duas resistências. Cada relé é alimentado com 5 volts, e possui um pino de controle conectado aos pinos 11 (D0) e 12 (D1) do XBee. Além disso, possuem outros três pinos: o pino comum, conectado à fase da tomada, e os pinos normalmente fechado (desconectado) e normalmente aberto (conectado à lâmpada). Quando o botão de acender a lâmpada é acionado, o pino digital do XBee referente à lâmpada desejada é energizado com 3.3 volts, acionando o pino de controle do relé, permitindo que a tensão de 110 volts da tomada chegue até a lâmpada acendendo-a.

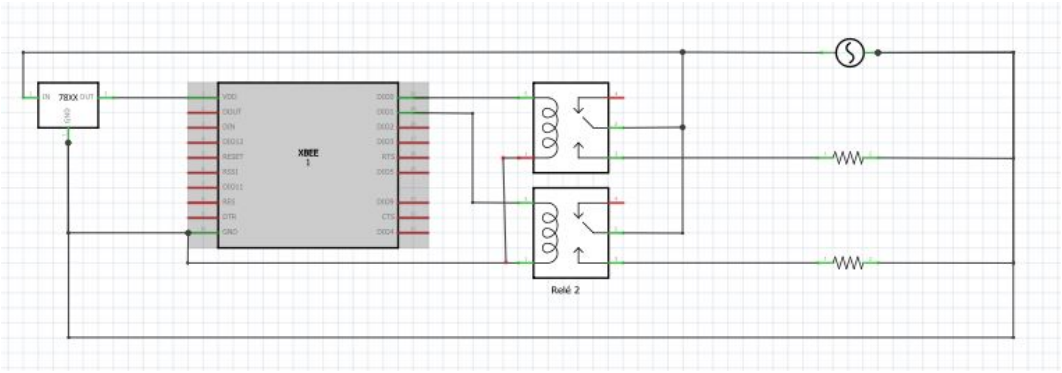

Figura 3.11: Esquema de montagem do módulo relé

3.4.2 Módulo atuador infravermelho

Os módulos IR atuam no controle de dispositivos que normalmente precisam de um controle remoto para controlá-los, como ar condicionados, ventiladores de teto, televisões, dvd-players, dentre outros. Neste projeto foram controlados apenas um ar condicionado e um ventilador de teto. Como mostrado na figura 3.12, este

módulo é composto por um arduino uno, um led infravermelho associado a uma resistência para regular a corrente e um XBee. A alimentação do XBee se dá pelo arduino, através de jumpers, utilizando os pinos de energia do arduino (3.3 Volts e ground), e sua comunicação serial com o arduino é realizada conectando-se os pinos Din e Dout aos pinos Tx e Rx do arduino respectivamente. Um pino digital do arduino é ligado ao led infravermelho através de um resistência de 10 ohm. Os XBees recebem os comandos vindos do módulo central e redireciona através da comunicação serial ao arduino, que através da sketch interpreta o comando e aciona os dispositivos finais a partir do led infravermelho.

O código para acionamento do led IR, disponível no A.2, e cujo fluxograma está representado na figura 3.13, é responsável por decodificar o frame API enviado pelo modúlo central, e enviar o respectivo sinal IR através do led ao dispositivo que se deseja controlar. Na etapa de inicialização, é realizada a inclusão das bibliotecas necessárias e declaração e definição das variáveis globais. Após a inicialização, o programa roda a função *setup*, responsável por iniciar a comunicação serial. Em seguida, será chamada a função *loop* que irá extrair do frame recebido pelo XBee apenas o comando desejado, redirecionando-o à função *rf_ data_ decoder*. Ao receber o comando, ela irá interpretar o comando, e chamará uma das duas funções responsáveis por comandar os dispositivos finais: *air_ conditioner* e *fan*. Nestas funções o comando recebido é convertido em um sinal infravermelho a ser enviado pelo led.

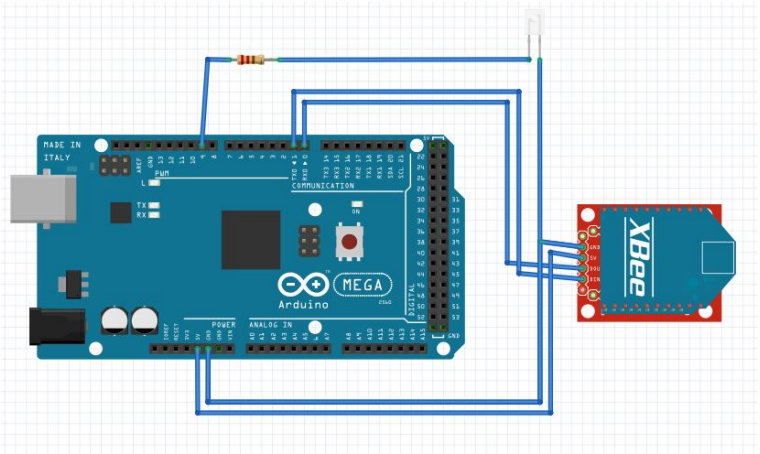

Figura 3.12: Esquema de montagem do módulo atuador infravermelho

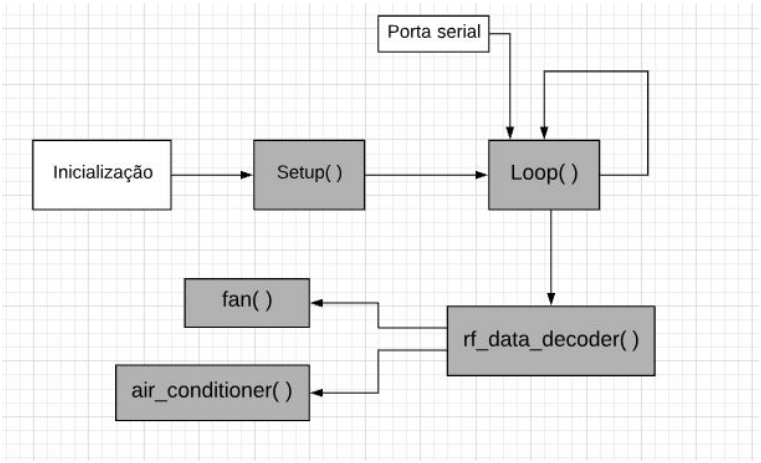

Figura 3.13: Fluxograma de funcionamento da sketch do módulo atuador infravermelho

A aquisição dos sinais infravermelhos do ar condicionado e do ventilador se deu a partir de uma outra sketch, disponível no apêndice . Através dessa sketch, é possível recriar sinais infravermelhos de qualquer protocolo. Isso é possível pois ao invés de decodificar o sinal recebido do led do controle remoto utilizado, ela armazena os intervalos de tempo em que o led está aceso ou apagado, permitindo assim recriar esse sinal posteriormente. Para realizar a aquisição dos sinais do controle remoto foi utilizado um arduino uno, um receptor de sinais infravermelhos (o modelo utilizado foi Pna4602m), e o controle remoto do qual deseja-se obter os sinais IR. Os valores negativos adquiridos na leitura do sinal do controle remoto são relativos aos intervalos em que o led estava apagado, ao passo que os positivos correspondem ao led aceso. É importante notar que os arrays contendo os sinais IR no código do apêndice A.2 devem possuir apenas valores positivos, logo, deve-se apagar todos os sinais '-' que antecipam os valores negativos dentro dos arrays.

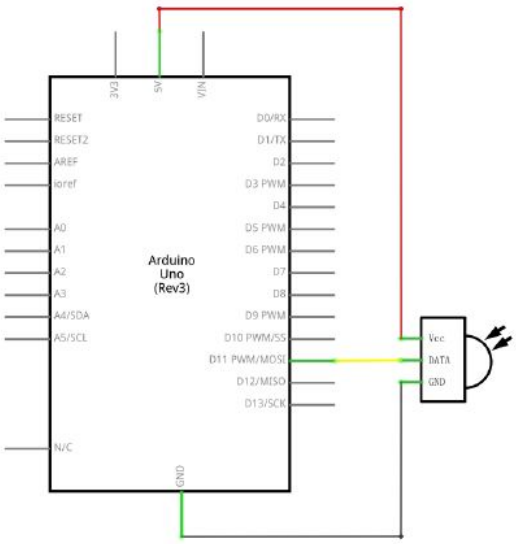

Figura 3.14: Esquema do circuito para recepção de sinais IR

3.5 Aplicativo

O aplicativo funciona como a interface entre usuário e sistema. Nele, o usuário dispõe de botões para acionamento dos dispositivos desejados, e displays que indicam o status desses dispositivos.

O código do programa, disponível no apêndice, é bem simples e o ambiente de desenvolvimento do aplicativo no MIT APP Inventor facilita a visualização do mesmo. O aplicativo está dividido em 5 janelas: screen1, Quarto, Lampadas, Ar Condicionado e Ventilador. Note que a janela inicial não pode ter o nome alterado, por isso chama-se screen1.

Os botões de comando presentes no aplicativo funcionam através da solicitação de um comando GET HTML ao servidor com uma URL definida como http://IPADDRESS/COMANDO, onde IPADDRESS é o endereço IP do servidor web, com a porta TCP inclusa no endereço, e COMANDO é o comando enviado ao módulo central, que neste projeto é composto por um identificador de início do código “Req”, seguido pelo comando de 8 caracteres, e um identificador de fim do comando “fim”. Por exemplo o comando ReqA/CSTF18fim é o comando de definir a temperatura do ar condicionado em 18 graus celsius. A lista com todos os comandos esta disponível no apêndice.

O código da janela screen1, disponível no apêndice 1, é bem simples e existem apenas duas ações disponíveis nesta janela: clicar no botão de voltar do celular ou

no botão de avançar. Clicar no botão de voltar do celular encerra o aplicativo, ao passo que clicar no botão de avançar define a variável "ip_address" com o endereço ip digitado na caixa de texto, e posteriormente chama a janela "Quarto".

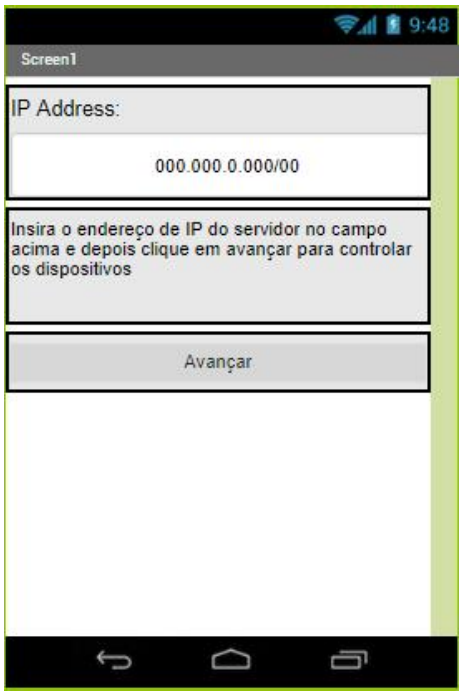

Figura 3.15: Janela inicial

Na janela "quarto", cujo código está pode ser lido no apêndice 2 estão disponíveis os dispositivos que podem ser controlador naquele ambiente, no caso deste projeto são: ventilador, lâmpadas e ar condicionado. Ao clicar em cada um dos botões ele chama a respectiva janela, enquanto que clicar no botão de voltar do celular fecha esta janela e chama a janela screen1 novamente.

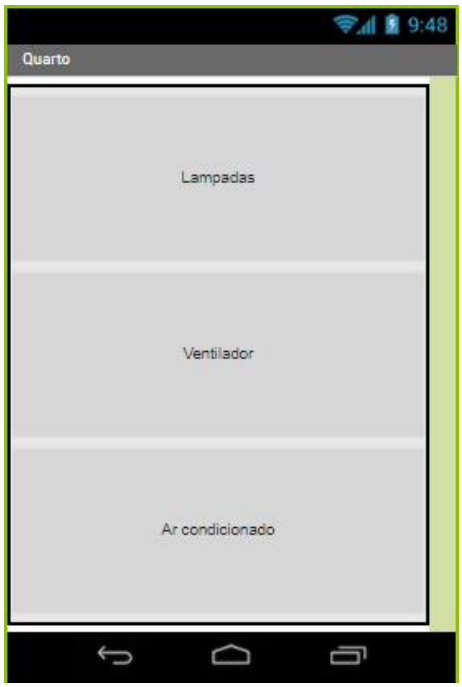

Figura 3.16: Janela do ambiente a ser controlado

Na janela "Lâmpadas", existem dois botões e um display para cada uma das duas lâmpadas que se pode controlar. Os botões de acender e apagar enviam os respectivos comandos ao módulo central. Este, executa as ações correspondentes e envia uma resposta html ao aplicativo, que atualiza o status das lâmpadas nos displays, como visto na figura 3.17.

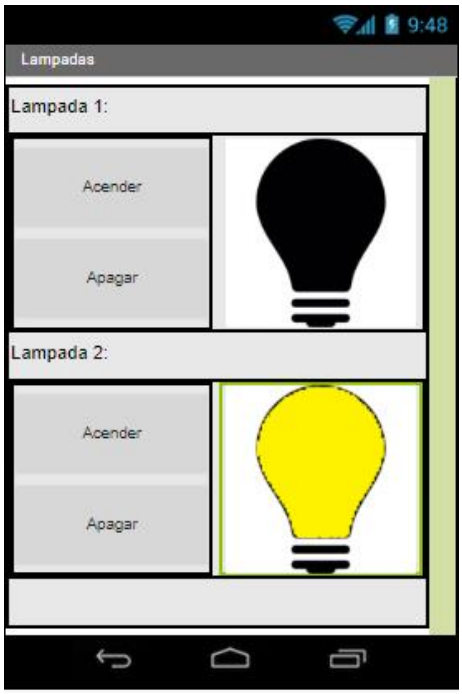

Figura 3.17: Janela de controle das lampadas

Na janela “Ventilador”, existem quatro botões de controle: Power, Reverse, Speed up e slow down. Cada um desses botões, ao ser pressionado, envia o respectivo comando ao módulo central, mas ao contrário da janela lâmpadas e ar_condicionado, elas não verificam a resposta html enviada pela central, visto que ela não possui displays de status do ventilador.

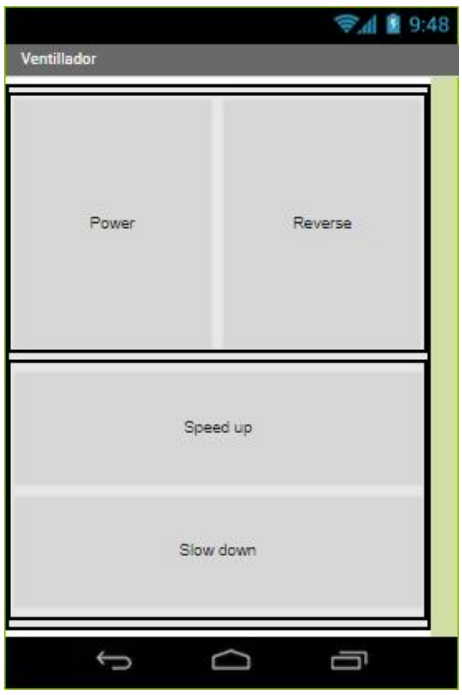

Figura 3.18: Janela de controle do ventilador

A janela “Ar_condicionado” é a que possui uma lógica de programação um pouco mais complexa. Ela possui os botões de ligar e desligar o ar condicionado, localizados na parte superior esquerda, os botões de seleção do modo de operação (ar condicionado ou desumidificador) e de movimento das palhetas, localizados na parte superior direita, e os botões de controle da temperatura e da vazão localizados na parte inferior da janela. Além disso, ela conta com dois displays localizados na faixa central da janela, uma para a temperatura a esquerda, e outra para a vazão ao lado direito. O display de temperatura varia entre 18 e 30 graus celsius, e conta com a opção “D”, quando o ar condicionado se encontra em modo de desumidificação. Já o display da vazão conta com apenas duas opções: vazão total e vazão parcial. Os botões de ligar, desligar e do movimento das pás tem uma lógica mais simples. Eles possuem um comando especifico, que são enviados à central quando esses botões são apertados. Os botões de seleção do modo de operação, verificam primeiramente o status da vazão atual, para posteriormente enviar o comando do modo de operação, mantendo a mesma vazão. Os botões de controle da temperatura, assim como os de controle da vazão, quando apertados, realizam a leitura da temperatura e vazão

atual em primeiro lugar, para depois enviar o comando adequado de temperatura e vazão ao módulo central.

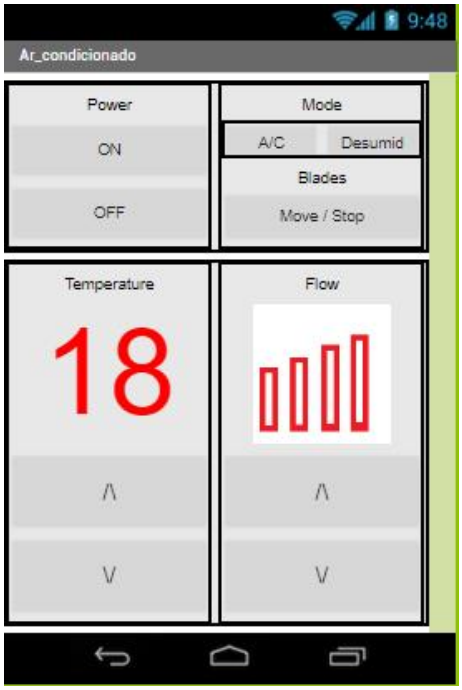

Figura 3.19: Janela de controle do ar condicionado

3.6 Video

O sistema de automação residencial foi implementado e testado, garantindo o funcionamento e a viabilização de utilização do mesmo para o propósito desejado. O video disponibilizado neste link (`https://www.youtube.com/watch?v=gLtuVMA7Bxo`) mostra o sistema implementado e em funcionamento.

Capítulo 4

Conclusões e trabalhos futuros

4.0.1 Resultados

Foi desenvolvido neste trabalho um sistema de automação residencial controlado por aplicativo em smartphone, que foi implementado utilizando a plataforma arduino e o módulo de rádio frequência XBee. Os resultados obtidos com este projeto foram bastante satisfatórios no que diz respeito ao funcionamento do mesmo, sendo capaz de realizar o acionamento dos módulos relé e do módulo infravermelho, possibilitando portanto o controle das lâmpadas, ventilador de teto e ar condicionado, apesar de algumas dificuldades e limitações que foram encontradas ao longo do desenvolvimento do projeto. O módulo de rádio frequencia XBee se sobressaiu como um dos pontos fortes deste projeto. Ele se mostrou muito confiável, cumprindo as especificações de alcance, e muito robusto quanto a comunicação, sem falhas ou perda de dados. Além disso é um módulo muito versátil, podendo funcionar em diversos modos de operação. A utilização do arduino para este projeto também se mostrou uma boa escolha, principalmente pela simplicidade de utilização, grande quantidade de material divulgado entre a comunidade de usuários dessa plataforma, e em relação ao custo. A utilização de outras plataformas, como por exemplo o Raspberry, porém, poderia melhorar o desempenho do sistema, devido a maior memória, possuir WIFI integrado e muitas outras funcionalidades. Entretanto, durante a realização deste projeto foram encontradas algumas dificuldades e limitações, mas soluções foram encontradas para viabilizar a conclusão e funcionamento do mesmo. Dentre as dificuldades, pode-se mencionar a utilização do módulo ENC28J60, um módulo ethernet de baixo custo, que foi adquirido na etapa inicial do projeto para permitir a conexão do módulo central à internet. Esse módulo era capaz de criar conexão à internet, mas não de manter a mesma estável por mais de cinco minutos, fazendo com que fosse necessário realizar um reset físico no arduino constantemente. Após muito tempo perdido tentando estabilizar o módulo com diversas bibliotecas, optou-se

pela utilização do shield ethernet, que na primeira tentativa já foi capaz de manter uma conexão estável por dias.

O sistema de automação implementado neste projeto, como dito anteriormente, possui uma arquitetura centralizada, em que apenas o módulo central é conectado à internet, e a comunicação entre os dispositivos do sistema é realizada através do protocolo ZigBee. Entretanto, existem outros tipos de arquitetura disponíveis para o desenvolvimento de um sistema de automação residencial. Como abordado na seção 1, é possível utilizar uma arquitetura horizontal, em que todos os dispositivos do sistema funcionem como um ponto de acesso à internet, como no conceito de internet das coisas. Para isso, porém, é necessário que todos esses dispositivos estejam dentro do alcance da rede WiFi utilizada. Nos testes realizados dentro de casa, por exemplo, os módulos XBee possuiam alcance dentro de toda a residência, ao passo que a rede WiFi era capaz de cobrir apenas 70% da área dela. Para utilização de uma arquitetura horizontal, seria necessária a inclusão de repetidores de sinal na residência para uma cobertura total.

4.0.2 Trabalhos futuros

O projeto que foi montado e implementado é um pouco mais simples que o projeto idealizado no início. Desejava-se, a princípio, complementar o acionamento dos dispositivos utilizados no projeto, e isso foi possível apenas para o ar condicionado e ventilador (dispositivos comandados pelo módulo atuador infravermelho). Para as lâmpadas, porém não foi possível utilizar o relé em paralelo com os interruptores devido a ausência de um sensor de corrente (ou de um regulador de tensão) conectado à fase da lâmpada por não possuí-los e devido à falta de tempo hábil para adquirí-los e implementá-los até a data determinada para conclusão deste trabalho. A figura 4.1 mostra o projeto inicialmente idealizado, que pode ser comparado ao implementado, representado pela figura 3.11 na página 35.

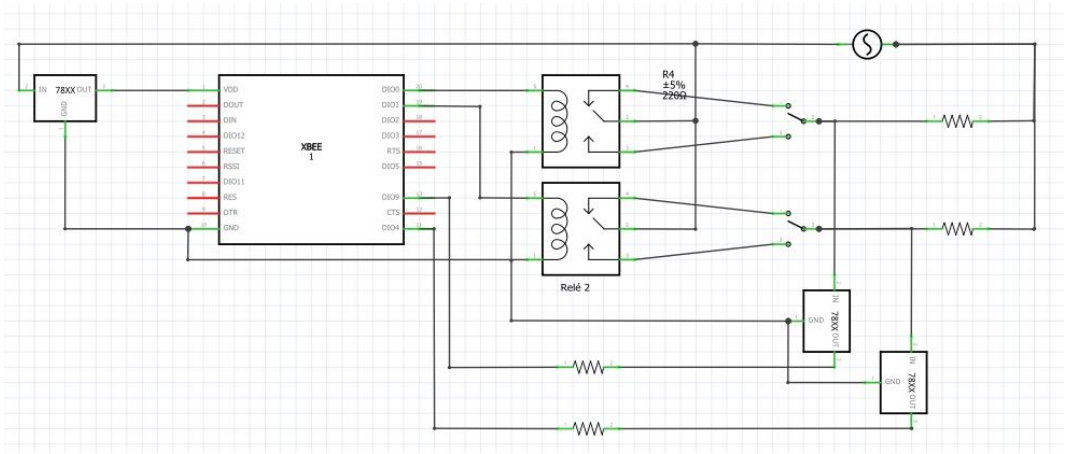

Figura 4.1: Esquema de montagem do módulo relé idealizado

A diferença entre os projetos se dá pela presença dos sensores de corrente, conectados entre a saída dos interruptores e o XBee. Sem o sensor e com a conexão threeway não é possível saber se a lâmpada se encontra ligada ou desligada, pois caso alguém acione mecanicamente as lâmpadas, essa ação não é reconhecida no módulo central, e um comando dado no aplicativo pode ter efeito contrário dependendo da posição do interruptor. Com os sensores, o módulo central pode efetuar uma leitura da porta digital do XBeel ligada ao sensor, permitindo obter o status atual da lâmpada antes de enviar o comando para o módulo relé, fazendo com que o comando realizado sempre seja igual ao enviado. Na realização do projeto, poucas seriam as mudanças necessárias com a adição dos sensores: seria necessário criar uma função no arduino do módulo central para criar frames API de solicitação remota de amostra da porta digital do XBee do módulo atuador relé (ZigBee IO Data Sample Rx Indicator - frame type 0x92) e efetuar uma pequena mudança na função send_relay(), para que ele enviasse o comando de ativação ou desativação do pino digital do XBee ligado ao relé de acordo com o status atual da lâmpada e do comando desejado. O módulo infravermelho também poderia receber algumas melhorias. No projeto testado, com o módulo comandando apenas o ventilador, o ar condicionado e o ventilador do quarto, a memória do arduino é suficiente se utilizado o modelo Mega. No modelo Uno, de apenas 2kb de sram, não há espaço para os 3kb utilizados em variáveis globais. Isso acontece devido ao grande tamanho de dados utilizados pelos arrays contendo os sinais infravermelhos. Há solução porém para este problema, é possível utilizar um cartao de memória SD contendo todos esses sinais, o que deixaria o programa com memoria SRAM disponível. Outra solução é utilizar um Arduino Mega, como de fato ocorreu no projeto (por não contar com dois arduinos uno). Em relação a trabalhos futuros, a aplicação do sistema desenvolvido neste projeto não se limita a automação residencial. Com pequenas modificações, este sistema poderia ser aprimorado para outras aplicações de smart buildings, como

também para diversas outras aplicações que necessitem de sensoriamento remoto e/ou acionamento remoto de atuadores em diversas áreas, principalmente em ambientes em que haja dificuldade de implementar uma rede com protocolo wifi, em que o protocolo ZigBee (que não depende de acesso a internet) seria mais eficiente, como no monitoramento e controle de irrigação em lavouras [2], monitoramento e controle de iluminação em grandes áreas externas, monitoramento e controle de temperatura de estufas, e também em ambientes que necessitem de mudanças frequentes na disposição física dos componentes, uma instalação temporária, como na automação de feiras de exposição e eventos.

Referências Bibliográficas

[1] LINS, V., MOURA, W. "DOMÓTICA: AUTOMAÇÃO RESIDENCIAL". Disponível em: <http://www.unibratec.edu.br/tecnologus/wp-content/uploads/2010/12/lins_moura.pdf>. [Acesso em 20 de julho de 2017].

[2] PODEROSO, F., SOBRAL, V., LIMA, R., et al. "Rede ZigBee Aplicada à Medição em Agricultura", *VIII Semetro*, 2009.

[3] EVANS, M., NOBLE, J., HOCHENBAUM, J. *Arduino em Ação*. Sâo Paulo, Novatec, 2013.

[4] DIGI. "Zigbee RF Modules User Guide". Disponível em: <https://www.digi.com/resources/documentation/digidocs/pdfs/90000976.pdf>. [Acesso em 20 de junho de 2017].

[5] RATHNAYAKA, A. J., POTDAR, V. "Evaluation of Wireless Home Automation Technologies", *5th IEEE International Conference on Digital Ecosystems and Technologies*, 2011.

[6] VARCHOLA, M., DRUTAROVSKÝ, M. "Zigbee Based Home Automation Wireless Sensor Network", *Acta Electrotechnica et Informatica*, v. 7, n. 4, 2007.

[7] JADHAV, P., CHAUDHARI, A., VAVALE, S. "Home Automation using ZigBee Protocol", *International Journal of Computer Science and Information Technologies*, v. 5, n. 2, 2014.

[8] C.GOMEZ, PARADELLS, J. "Wireless home automation networks: A survey of architectures and technologies", *IEEE Communications Magazine*, v. 48, n. 6, 2010.

[9] MIT. "Making Mobile Apps with App Inventor". Disponível em: <http://appinventor.mit.edu/explore/ai2/beginner-videos.html>. [Acesso em 2 de outubro de 2017].

[10] ANALYSIR. “Air conditioners: Recording long infrared Remote control signals with Arduino”.
https://www.analysir.com/blog/2014/03/19/air-conditioners-problems-recording-long-infrared-remote-control-signals-arduino/. [Acesso em 15 de maio de 2018].

Apêndice A

Códigos Arduino

A.1 Módulo central

```
// ——-Incluir bibliotecas———
#include <SPI.h>
#include <Ethernet.h>

    // Strings de leitura de comandos
const int size_buffer = 20;
char * buf_ptr;
char * cmd_ptr;
bool start_index_found = false;
int cmd_start_index;
int cmd_end_index;
char buf[size_buffer]={};
//String readString;
const int lim_readString = 50;
char cmd[12]={};

    // Configurações de IP
// MAC Adress deve ser único na rede

    byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};

    // Endereço de IP utilizado pelo arduino

    IPAddress ip(192, 168, 0, 103);
```

```
    // Cria um servidor para ouvir uma porta especifica
// (porta 80 para http)

    EthernetServer web_server(80);

    void setup() {

    Serial.begin(9600); //XBee serial also set to 9600 on digi xctu
Serial.println("Conexão serial estabelecida!");
Serial.println("Iniciando sistema...");

    // Inicializar biblioteca Ethernet web_server_init();

    }

    void web_server_init(){

    // start the Ethernet connection and the server:
Ethernet.begin(mac, ip);
web_server.begin();
Serial.println("Inicializando servidor web");
Serial.print("IP local do servidor: ");
Serial.println(Ethernet.localIP());
Serial.println("Servidor web inicializado com sucesso!");
Serial.println(-————————————————————————");

    }

    void send_app(EthernetClient sendAns, char* ans){

    const int size_app_ans = 12;
char app_ans[size_app_ans]={};
Serial.println("Send app na area");

    for (int i=0; i<size_app_ans; i++){
app_ans[i] = ans[i];
}

    Serial.println(app_ans);
```

```
    sendAns.println("HTTP/1.1 200 OK");
sendAns.println("Content-Type: text/html");
sendAns.println("Connection:  close"); // the connection will be closed after
completion of the response
sendAns.println();
sendAns.print(app_ans);
sendAns.println(«/html>");
return;

    }

    void set_relay(char* dev, char* cmd_l){

    long checksum;
Serial.println("Set_relay");

    Serial.write(0x7E); //Start Byte

    Serial.write((byte)0x0); // lenght MSB
Serial.write(0x10); // lenght LSB

    Serial.write(0x17); //0x17: Frame ID pra envio de comando AT
Serial.write((byte)0x0); //Frame ID (no reply needed)

    Serial.write(0x00); // Inicio do endereço de 64 bits
Serial.write(0x13);
Serial.write(0xA2);
Serial.write(0x00);
Serial.write(0x40);
Serial.write(0x9B);
Serial.write(0x12);
Serial.write(0x5F); // Fim do endereço de 64 bits

    Serial.write(0xFF); // Destination newtwork
Serial.write(0xFE); // (set to 0xFFFE se não souber)
Serial.write(0x02); // Set to 0x02 to apply these changes
if (dev == "L01"){
Serial.write('D'); // AT Command: D0
```

```
Serial.write('0');
checksum = 0x17 + 0x0 + 0x00 + 0x13 + 0xA2 + 0x00 + 0x40 + 0x9B + 0x12 +
0x5F + 0xFF + 0xFE + 0x02 + 'D' + '0';
}
else if (dev == "L02"){
Serial.write('D'); // AT Command: D1
Serial.write('1');
checksum = 0x17 + 0x0 + 0x00 + 0x13 + 0xA2 + 0x00 + 0x40 + 0x9B + 0x12 +
0x5F + 0xFF + 0xFE + 0x02 + 'D' + '1';
}

    if (cmd_1 == "on"){
Serial.write(0x05);
checksum = checksum + 0x05;
}
else if (cmd_1 == "off"){
Serial.write(0x04);
checksum = checksum + 0x04;
}
Serial.write(0xFF - (checksum & 0xFF)); //Checksum
Serial.println();

    }

    void set_ir(char *cmd){

    //Serial.println("entrou na funcao xbee on");
char byte_1 = cmd[0];
char byte_2 = cmd[1];
char byte_3 = cmd[2];
char byte_4 = cmd[3];
char byte_5 = cmd[4];
char byte_6 = cmd[5];
char byte_7 = cmd[6];
char byte_8 = cmd[7];

    // Start delimiter (Start byte)
char offset_0 = 0x7E; //Start Byte
Serial.write(offset_0);
```

```
    // Lenght (bytes between lenght and checksum)
// 8 bytes de payload -> 19 bytes (0x13)
char offset_1 = 0x00; // 1
char offset_2 = 0x16; // 2
Serial.write(offset_1);
Serial.write(offset_2);

    // 3 Frame type: Zigbee Transmit Request (0x10)

    char offset_3 = 0x10; // 3
Serial.write(offset_3);

    // 4 Frame ID : No response is sent (0x00)

    char offset_4 = 0x01; // 4
Serial.write(offset_4);

    // 64-bit Destination Address: 0013A200409B126B
char offset_5 = 0x00; // 5
char offset_6 = 0x13; // 6
char offset_7 = 0xA2; // 7
char offset_8 = 0x00; // 8
char offset_9 = 0x40; // 9
char offset_10 = 0x9B; // 10
char offset_11 = 0x12; // 11
char offset_12 = 0x6B; // 12 0x5F

    Serial.write(offset_5);
Serial.write(offset_6);
Serial.write(offset_7);
Serial.write(offset_8);
Serial.write(offset_9);
Serial.write(offset_10);
Serial.write(offset_11);
Serial.write(offset_12);

    // 16-bit Destinantion Network: FFFE (broadcast)
```

```
    char offset_13 = 0xFF; // 13
char offset_14 = 0xFE; // 14
Serial.write(offset_13);
Serial.write(offset_14);

    // Broadcast Radius: 0 (maximum hops value)

    char offset_15 = 0x00; // 15
Serial.write(offset_15);

    // Options: 0 none
    char offset_16 = 0x00; // 16
Serial.write(offset_16);

    // RF Data: 5 bytes de comando para o XBee

    char offset_17 = byte_1;
char offset_18 = byte_2;
char offset_19 = byte_3;
char offset_20 = byte_4;
char offset_21 = byte_5;
char offset_22 = byte_6;
char offset_23 = byte_7;
char offset_24 = byte_8;

    Serial.write(offset_17);
Serial.write(offset_18);
Serial.write(offset_19);
Serial.write(offset_20);
Serial.write(offset_21);
Serial.write(offset_22);
Serial.write(offset_23);
Serial.write(offset_24);

    // Checksum (0xFF - the 8 bit sum of bytes from offset 3 to this offset)
char checksum = offset_3 + offset_4 + offset_5 + offset_6 + offset_7 + offset_8
+ offset_9 + offset_10 + offset_11 + offset_12 + offset_13 + offset_14 + off-
set_15 + offset_16 + offset_17 + offset_18 + offset_19 + offset_20 + offset_21
+ offset_22 + offset_23 + offset_24 ;
```

```
//char checksum = offset_3 + offset_4 + offset_5 + offset_6 + offset_7 + off-
set_8 + offset_9 + offset_10 + offset_11 + offset_12 + offset_13 + offset_14 +
offset_15 + offset_16 + offset_17 + offset_18 + offset_19 + offset_20 + offset_21;

    char offset_25 = 0xFF - (checksum & 0xFF);
Serial.write(offset_25); //22
Serial.println();

    }

    void redirect_commands(EthernetClient sendClient, char *received_cmd){

    const int size_app_cmd = 9;
const int size_feedback_cmd = size_app_cmd + 3;
char app_cmd[size_app_cmd]={};

    for (int i=0; i<size_app_cmd; i++){
app_cmd[i] = received_cmd[i];
}

    // COMANDOS DAS LAMPADAS ————————————————————
cmd_ptr = strstr(app_cmd, "RELL01OF");
if (cmd_ptr != 0){

    char feedback_cmd[size_feedback_cmd] = "AnsRELL01OF";
send_app(sendClient, feedback_cmd);

    set_relay("L01", "off");

    // Resetando variaveis
cmd_ptr = 0;

    for( int k = 0;
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
```

```
k < sizeof(received_cmd);
k++ ){
received_cmd[k] = (char)0;
}
}

    cmd_ptr = strstr(app_cmd, "RELL01ON");
if (cmd_ptr != 0){

    char feedback_cmd[size_feedback_cmd] = "AnsRELL01ON";
send_app(sendClient, feedback_cmd);

    set_relay("L01", "on");

    // Resetando variaveis
cmd_ptr = 0;

    for( int k = 0;
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(received_cmd);
k++ ){
received_cmd[k] = (char)0;
}
}

    cmd_ptr = strstr(app_cmd, "RELL02OF");
if (cmd_ptr != 0){

    char feedback_cmd[size_feedback_cmd] = "AnsRELL02OF";
send_app(sendClient, feedback_cmd);

    set_relay("L02", "off");

    // Resetando variaveis
```

```
cmd_ptr = 0;

    for( int k = 0;
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(received_cmd);
k++ ){
received_cmd[k] = (char)0;
}
}

    cmd_ptr = strstr(app_cmd, "RELL02ON");
if (cmd_ptr != 0){

    char feedback_cmd[size_feedback_cmd] = "AnsRELL02ON";
send_app(sendClient, feedback_cmd);

    set_relay("L02","on");

    // Resetando variaveis
cmd_ptr = 0;

    for( int k = 0;
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(received_cmd);
k++ ){
received_cmd[k] = (char)0;
}
}
```

```
    // COMANDOS DO VENTILADOR ----------------------------
cmd_ptr = strstr(app_cmd, "FANPOWER");
if (cmd_ptr != 0){

    char feedback_cmd[size_feedback_cmd] = "AnsFANPOWER";
send_app(sendClient, feedback_cmd);

    set_ir("fanpower");

    // Resetando variaveis
cmd_ptr = 0;

    for( int k = 0;
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(received_cmd);
k++ ){
received_cmd[k] = (char)0;
}

    return;
}

    cmd_ptr = strstr(app_cmd, "FANREVER");
if (cmd_ptr != 0){

    char feedback_cmd[size_feedback_cmd] = "AnsFANREVER";
send_app(sendClient, feedback_cmd);

    set_ir("fanrever");

    // Resetando variaveis
cmd_ptr = 0;

    for( int k = 0;
```

```
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(received_cmd);
k++ ){
received_cmd[k] = (char)0;
}

    return;
}

    cmd_ptr = strstr(app_cmd, "FANFASTR");
if (cmd_ptr != 0){

    char feedback_cmd[size_feedback_cmd] = "AnsFANFASTR";
send_app(sendClient, feedback_cmd);

    set_ir("fanfastr");

    // Resetando variaveis
cmd_ptr = 0;

    for( int k = 0;
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(received_cmd);
k++ ){
received_cmd[k] = (char)0;
}

    return;
}
```

```
    cmd_ptr = strstr(app_cmd, "FANSLOWR");
if (cmd_ptr != 0){

    char feedback_cmd[size_feedback_cmd] = "AnsFANSLOWR";
send_app(sendClient, feedback_cmd);

    set_ir("fanslowr");

    // Resetando variaveis
cmd_ptr = 0;

    for( int k = 0;
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(received_cmd);
k++ ){
received_cmd[k] = (char)0;
}

    return;
}

    //COMANDOS DO AR CONDICIONADO ————————————————————

    // LIGAR cmd_ptr = strstr(app_cmd, "A/CONF18");
if (cmd_ptr != 0){

    Serial.println("LIGANDO AR CONDICIONADO...");

    char feedback_cmd[size_feedback_cmd] = "AnsA/CONF18";
send_app(sendClient, feedback_cmd);

    set_ir("a/conF18");
// Resetando variaveis
```

```
feedback_cmd[size_app_cmd] = {};

    for( int k = 0;
k < sizeof(received_cmd);
k++ ){
received_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(app_cmd);
k++ ){
app_cmd[k] = (char)0;
}

    return;
}

    // DESLIGAR
cmd_ptr = strstr(app_cmd, "A/COF___");
if (cmd_ptr != 0){

    Serial.println("DESLIGANDO AR CONDICIONADO...");
char feedback_cmd[size_feedback_cmd] = "AnsA/COF___";
send_app(sendClient, feedback_cmd);

    set_ir("a/cof___");

    // Resetando variaveis
for( int k = 0;
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(app_cmd);
k++ ){
app_cmd[k] = (char)0;
}
```

```
    for( int k = 0;
k < sizeof(received_cmd);
k++ ){
received_cmd[k] = (char)0;
}

    return;
}

    // Movimento das palhetas
cmd_ptr = strstr(app_cmd, "A/CBM___");
if (cmd_ptr != 0){

    Serial.println("Blades motion...");
char feedback_cmd[size_feedback_cmd] = "AnsA/CBM___";
send_app(sendClient, feedback_cmd);

    set_ir("a/cbm___");
// Resetando variaveis
for( int k = 0;
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(app_cmd);
k++ ){
app_cmd[k] = (char)0;
}

    return;
}

    //TEMPERATURA
cmd_ptr = strstr(app_cmd, "A/CST");
if (cmd_ptr != 0){
```

```
    // Lendo valores do fluxo e temperatura
char flow_d = app_cmd[size_app_cmd-4];
char temp_d = app_cmd[size_app_cmd-3];
char temp_u = app_cmd[size_app_cmd-2];

    // Construindo redirecionamento xbee
char xbee_cmd[size_app_cmd] = "a/cst";
xbee_cmd[7] = temp_u;
xbee_cmd[6] = temp_d;
xbee_cmd[5] = flow_d;

    Serial.print("IR cmd = ");
Serial.println(xbee_cmd);

    set_ir(xbee_cmd); // redirecionamento xbee

    // Construindo redirecionamento app
char feedback_cmd[size_feedback_cmd] = "AnsA/CST_";
feedback_cmd[size_feedback_cmd - 3] = temp_d;
feedback_cmd[size_feedback_cmd - 2] = temp_u;

    Serial.print("Resposta ao app = ");
Serial.println(feedback_cmd);

    send_app(sendClient, feedback_cmd);

    // Resetando variaveis for( int k = 0;
k < sizeof(feedback_cmd)+1;
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(xbee_cmd);
k++ ){
xbee_cmd[k] = (char)0;
}

    for( int k = 0;
```

```
k < sizeof(app_cmd);
k++ ){
app_cmd[k] = (char)0;
}

    flow_d = ;
temp_d = ;
temp_u = ;
return;
}

    //Fluxo
cmd_ptr = strstr(app_cmd, "A/CSF");
if (cmd_ptr != 0){

    // Lendo valores do fluxo e temperatura
char flow_d = app_cmd[size_app_cmd-4];
char temp_d = app_cmd[size_app_cmd-3];
char temp_u = app_cmd[size_app_cmd-2];

    // Construindo redirecionamento xbee
char xbee_cmd[size_app_cmd] = "a/csf";
xbee_cmd[7] = temp_u;
xbee_cmd[6] = temp_d;
xbee_cmd[5] = flow_d;

    Serial.print("IR cmd = ");
Serial.println(xbee_cmd);

    set_ir(xbee_cmd); // redirecionamento xbee

    // Construindo redirecionamento app
char feedback_cmd[size_feedback_cmd] = "AnsA/CSF";
feedback_cmd[size_feedback_cmd - 4] = flow_d;
feedback_cmd[size_feedback_cmd - 3] = 'T';
feedback_cmd[size_feedback_cmd - 2] = 'T';

    Serial.print("Resposta ao app = ");
Serial.println(feedback_cmd);
```

```
    send_app(sendClient, feedback_cmd);

    // Resetando variaveis for( int k = 0;
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(xbee_cmd);
k++ ){
xbee_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(app_cmd);
k++ ){
app_cmd[k] = (char)0;
}

    flow_d = ;
temp_d = ;
temp_u = ;
return;
}

    // Desumidificador
cmd_ptr = strstr(app_cmd, "A/CDE");
if (cmd_ptr != 0){

    // Lendo valores do fluxo e temperatura
char flow_d = app_cmd[size_app_cmd-4];

    Serial.print("Flow: ");
Serial.println(flow_d);

    // Construindo redirecionamento xbee
char xbee_cmd[size_app_cmd] = "a/cde";
```

```
xbee_cmd[7] = 'T';
xbee_cmd[6] = 'T';
xbee_cmd[5] = flow_d;

    Serial.print("IR cmd = ");
Serial.println(xbee_cmd);

    set_ir(xbee_cmd); // redirecionamento xbee

    // Construindo redirecionamento app
char feedback_cmd[size_feedback_cmd] = "AnsA/CDE";
feedback_cmd[size_feedback_cmd - 4] = flow_d;
feedback_cmd[size_feedback_cmd - 3] = 'T';
feedback_cmd[size_feedback_cmd - 2] = 'T';

    Serial.print("Resposta ao app = ");
Serial.println(feedback_cmd);

    send_app(sendClient, feedback_cmd);

    // Resetando variaveis for( int k = 0;
k < sizeof(feedback_cmd);
k++ ){
feedback_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(xbee_cmd);
k++ ){
xbee_cmd[k] = (char)0;
}

    for( int k = 0;
k < sizeof(app_cmd);
k++ ){
app_cmd[k] = (char)0;
}

    flow_d = ;
```

```
return;
}

    delay(1);
}

    void loop(){

    // listen for incoming clients
EthernetClient client = web_server.available();

    if (client) {

    int i = 0;
while (client.connected()) //Enquanto conectado
{

    char c = client.read();

    //if (readString.length() < len_string_buf){
if (strlen(buf) < size_buffer){
// armazena os caracteres para string
buf[i] = c;
//Serial.println(buf);
}

    buf_ptr = strstr(buf,"Req");
if (buf_ptr != 0 && start_index_found == false){

    cmd_start_index = i;
start_index_found = true;
//Serial.print("cmd_start_index = ");
//Serial.println(cmd_start_index);
}

    buf_ptr = strstr(buf,"fim");
if ( buf_ptr != 0) {

    cmd_end_index = i;
```

```
//Serial.print("cmd_end_index = ");
//Serial.println(cmd_end_index);

    int k = 0;
for (int j = cmd_start_index + 1;
j < cmd_end_index - 2;
j++){
cmd[k] = buf[j];
k++;
}
k=0;

    //Serial.print("Cmd = ");
Serial.println(cmd);

    redirect_commands(client, cmd);
client.stop();

    for( k = 0;
k < sizeof(buf);
k++ ){
buf[k] = (char)0;
}

    cmd_start_index = 0;
cmd_end_index = 0;
buf_ptr = 0;
start_index_found = false;
return;
}

    i++;

    }
}
}
```

A.2 Módulo atuador infravermelho

```
#include <IRremote.h>
    // Declaração de objetos e variaveis
IRsend irsend; //pino 9 no MEGA

    // Strings de leitura dos frames API
char * ptr;
const int buf_Size = 25;
char buf[buf_Size]={};
const int rf_data_size = 8;
char rf_data[rf_data_size];

    void setup() {

    //Begin serial connection for debugging
Serial.begin(9600);
//Begin serial connection with XBee
Serial1.begin(9600);
Serial.print("Setup finalizado!");
Serial.println();
}

    // Definição dos sinais infravermelhos do ventilador
const int fan_ir_lenght = 47;

    unsigned int power[fan_ir_lenght]= {1280, 416, 1272, 416, 1272, 424, 412,
1256, 412, 1264, 1272, 424, 412, 1260, 412, 1260, 1272, 416, 412, 1260, 412, 1256,
468, 35004, 1276, 416, 1272, 416, 1276, 420, 412, 1256, 416, 1260, 1276, 424, 412,
1260, 412, 1260, 1272, 416, 412, 1260, 412, 1288, 440};

    unsigned int speed_up[fan_ir_lenght]= {1276, 416, 1272, 416, 1276, 420, 412,
1260, 412, 1260, 1276, 420, 416, 1256, 412, 1260, 412, 1264, 408, 1260, 1276, 416,
464, 34996, 1276, 416, 1272, 416, 1272, 424, 412, 1260, 412, 1260, 1272, 424, 412,
1260, 412, 1260, 412, 1260, 412, 1260, 1276, 412, 468};

    unsigned int slow_down[fan_ir_lenght]= {1276, 416, 1272, 416, 1272, 424, 412,
1260, 408, 1284, 1252, 424, 412, 1280, 392, 1256, 412, 1264, 1272, 416, 412, 1280,
444, 34996, 1272, 416, 1276, 416, 1272, 424, 408, 1284, 388, 1260, 1276, 424, 412,
```

1260, 412, 1260, 412, 1264, 1272, 416, 412, 1280, 444};

unsigned int reverse[fan_ir_lenght]= {1276, 416, 1276, 416, 1272, 420, 416, 1256, 412, 1260, 1276, 424, 1276, 412, 412, 1260, 412, 1260, 412, 1260, 412, 1260, 468, 34996, 1276, 416, 1276, 416, 1272, 424, 408, 1260, 412, 1260, 1276, 428, 1272, 416, 412, 1260, 412, 1260, 412, 1256, 416, 1256, 472 };

// Definição dos sinais infravermelhos do ar condicionado
const int ac_ir_lenght = 59;

unsigned int off[ac_ir_lenght] = {8680, 4048, 528, 1516, 496, 524, 472, 540, 472, 544, 472, 1564, 500, 512, 472, 540, 500, 520, 472, 1556, 476, 1560, 472, 544, 500, 516, 472, 540, 476, 544, 472, 544, 476, 536, 476, 540, 500, 516, 472, 548, 472, 540, 476, 540, 496, 1540, 500, 512, 472, 1560, 472, 548, 500, 512, 472, 544, 476, 1556, 500};

unsigned int on_18_[ac_ir_lenght] = {8672, 4064, 528, 1520, 468, 548, 472, 540, 476, 544, 468, 1560, 472, 548, 468, 544, 472, 544, 472, 544, 472, 544, 472, 544, 472, 544, 496, 520, 472, 544, 472, 544, 476, 540, 496, 524, 472, 540, 472, 1560, 472, 1560, 472, 544, 472, 1560, 472, 544, 476, 540, 472, 544, 472, 1556, 476, 1560, 472, 1560, 472};

unsigned int temp_18_FULL_[ac_ir_lenght] = {8676, 4056, 528, 1516, 472, 548, 476, 536, 472, 544, 472, 1560, 500, 520, 500, 512, 472, 540, 476, 544, 500, 516, 496, 520, 472, 544, 472, 1556, 476, 544, 472, 544, 472, 544, 500, 516, 500, 516, 472, 1560, 472, 1560, 472, 544, 472, 1556, 472, 548, 472, 540, 472, 1564, 472, 1556, 472, 1560, 472, 1560, 472 };

unsigned int temp_19_FULL_[ac_ir_lenght] = {8672, 4060, 532, 1516, 472, 544, 476, 540, 496, 520, 472, 1564, 500, 512, 472, 544, 472, 544, 472, 544, 472, 540, 476, 540, 476, 544, 472, 1560, 472, 540, 472, 544, 472, 552, 496, 512, 472, 1560, 472, 548, 472, 540, 472, 544, 472, 1564, 500, 512, 472, 544, 472, 544, 472, 548, 496, 520, 472, 540, 472};

unsigned int temp_20_FULL_[ac_ir_lenght] = {8676, 4056, 528, 1516, 472, 544, 500, 516, 476, 540, 472, 1560, 504, 512, 472, 544, 496, 520, 472, 548, 500, 512, 472, 544, 472, 540, 472, 1560, 476, 544, 472, 544, 472, 544, 504, 508, 476, 1556, 496, 524, 500, 1528, 476, 540, 472, 1560, 472, 544, 472, 544, 472, 548, 500, 512, 476, 540, 472, 1564, 504};

unsigned int temp_21_FULL_[ac_ir_lenght] = {8676, 4056, 528, 1516, 472, 544, 500, 516, 472, 544, 472, 1560, 500, 516, 472, 544, 472, 544, 472, 548, 500, 512, 472, 544, 472, 544, 484, 1544, 476, 544, 472, 540, 476, 548, 500, 508, 500, 1536, 468, 1564, 468, 544, 472, 544, 472, 1560, 472, 544, 472, 548, 472, 540, 500, 516, 472, 1560, 472, 548, 500};

unsigned int temp_22_FULL_[ac_ir_lenght] = {8676, 4044, 532, 1512, 472, 548, 500, 516, 472, 540, 500, 1536, 500, 512, 472, 548, 500, 512, 472, 548, 500, 512, 472, 544, 476, 540, 504, 1528, 472, 548, 500, 512, 496, 520, 476, 540, 472, 1560, 472, 1560, 472, 1556, 476, 544, 504, 1528, 472, 540, 476, 548, 500, 512, 472, 544, 472, 1560, 500, 1528, 472};

unsigned int temp_23_FULL_[ac_ir_lenght] = {8676, 4056, 528, 1516, 472, 544, 476, 544, 468, 544, 472, 1552, 508, 516, 472, 544, 472, 544, 468, 552, 500, 512, 472, 544, 472, 544, 468, 1560, 476, 540, 476, 540, 476, 544, 500, 1528, 472, 544, 472, 548, 500, 516, 472, 544, 468, 1564, 500, 512, 472, 544, 472, 544, 472, 1564, 468, 544, 472, 544, 472};

unsigned int temp_24_FULL_[ac_ir_lenght] = {8676, 4052, 524, 1516, 476, 548, 500, 512, 472, 544, 468, 1568, 500, 512, 472, 544, 472, 540, 472, 552, 500, 508, 496, 524, 472, 540, 472, 1564, 472, 540, 472, 544, 472, 548, 500, 1528, 472, 544, 472, 548, 472, 1556, 472, 544, 472, 1560, 472, 544, 472, 544, 472, 544, 476, 1552, 476, 544, 472, 1560, 472};

unsigned int temp_25_FULL_[ac_ir_lenght] = {8676, 4060, 532, 1516, 468, 548, 500, 512, 500, 516, 472, 1568, 500, 508, 500, 516, 472, 544, 472, 544, 480, 536, 496, 520, 472, 544, 472, 1560, 472, 544, 472, 544, 472, 548, 500, 1524, 500, 520, 472, 1560, 500, 516, 472, 540, 476, 1556, 476, 540, 500, 516, 476, 544, 504, 1528, 472, 1560, 504, 508, 500};

unsigned int temp_26_FULL_[ac_ir_lenght] = {8680, 4056, 532, 1516, 472, 544, 500, 516, 496, 520, 472, 1564, 500, 508, 504, 512, 472, 548, 468, 548, 496, 520, 488, 524, 472, 544, 476, 1556, 476, 540, 472, 544, 472, 548, 504, 1524, 496, 520, 472, 1560, 500, 1528, 476, 544, 504, 1528, 472, 544, 472, 544, 504, 508, 500, 1536, 472, 1560, 496, 1532, 472};

unsigned int temp_27_FULL_[ac_ir_lenght] = {8676, 4060, 532, 1508, 476, 548, 500, 512, 500, 516, 472, 1564, 500, 512, 500, 516, 472, 544, 500, 516, 500, 516, 496, 520, 472, 544, 496, 1536, 472, 544, 472, 540, 476, 544, 504, 1524, 500, 1536,

504, 512, 500, 512, 500, 516, 500, 1532, 500, 516, 496, 520, 472, 1560, 500, 516, 472, 544, 472, 548, 504};

unsigned int temp_28_FULL_[ac_ir_lenght] = {8676, 4048, 532, 1512, 476, 548, 500, 512, 472, 540, 472, 1564, 500, 512, 496, 524, 472, 540, 472, 548, 500, 516, 468, 544, 472, 544, 472, 1560, 472, 544, 472, 548, 500, 516, 496, 1532, 472, 1564, 500, 512, 472, 1560, 472, 544, 476, 1552, 476, 544, 472, 544, 504, 1524, 476, 540, 476, 544, 500, 1532, 468};

unsigned int temp_29_FULL_[ac_ir_lenght] = {8648, 4088, 532, 1516, 472, 544, 504, 512, 472, 544, 472, 1560, 500, 512, 476, 540, 476, 540, 472, 548, 504, 508, 496, 520, 492, 524, 472, 1560, 472, 544, 476, 540, 496, 524, 500, 1528, 472, 1560, 504, 1528, 472, 548, 496, 516, 504, 1524, 476, 540, 492, 528, 504, 1524, 472, 544, 472, 1564, 468, 548, 472 };

unsigned int temp_30_FULL_[ac_ir_lenght] = {8676, 4052, 528, 1516, 476, 544, 500, 512, 472, 544, 472, 1564, 500, 512, 472, 544, 476, 540, 472, 548, 472, 540, 472, 544, 472, 544, 472, 1560, 472, 544, 472, 548, 500, 512, 472, 1560, 472, 1560, 500, 1532, 472, 1560, 472, 540, 472, 1564, 472, 544, 472, 544, 472, 1556, 472, 548, 472, 1556, 472, 1560, 476};

unsigned int desumid_HALF_[ac_ir_lenght] = {8648, 4088, 528, 1516, 472, 544, 476, 540, 472, 544, 472, 1564, 472, 540, 472, 544, 472, 544, 472, 544, 504, 512, 496, 520, 472, 544, 472, 1560, 472, 544, 472, 540, 500, 1536, 472, 1560, 468, 548, 500, 516, 472, 540, 472, 548, 472, 548, 500, 512, 472, 544, 472, 540, 472, 548, 472, 540, 472, 1560, 472};

unsigned int desumid_FULL_[ac_ir_lenght] = {8648, 4088, 528, 1520, 472, 544, 476, 540, 472, 540, 472, 1564, 500, 516, 472, 544, 472, 544, 472, 544, 472, 540, 472, 544, 476, 540, 472, 1560, 476, 544, 472, 540, 472, 1560, 476, 1560, 472, 544, 472, 540, 472, 544, 472, 544, 472, 1560, 476, 540, 472, 544, 472, 548, 472, 1556, 472, 544, 472, 1560, 472};

unsigned int blades_motion_[ac_ir_lenght] = {8680, 4056, 528, 1520, 500, 516, 500, 512, 472, 544, 472, 1564, 500, 512, 496, 520, 472, 544, 472, 544, 504, 512, 472, 544, 472, 1564, 500, 512, 472, 548, 500, 512, 472, 544, 504, 512, 472, 544, 472, 540, 472, 548, 504, 512, 472, 540, 476, 544, 472, 544, 504, 512, 468, 548, 472, 544, 472, 1556, 476};

```
    // função que aciona o ventilador
    void fan(int button){

    if (button == 1){
Serial.println("Power button pressed...");
irsend.sendRaw(power, fan_ir_lenght, 38);
} else if (button == 2){
Serial.println("Slow down button pressed...");
irsend.sendRaw(slow_down, fan_ir_lenght, 38);
} else if (button == 3){
Serial.println("Speed up button pressed...");
irsend.sendRaw(speed_up, fan_ir_lenght, 38);
} else if (button == 4){
Serial.println("Reverse button pressed...");
irsend.sendRaw(reverse, fan_ir_lenght, 38);
} else {
Serial.print(button);
Serial.println("Nenhum botão acionado");
}
    }
    void air_conditioner(int mode,char f_d, char t_d, char t_u){

    char temp[2] = {t_d,t_u};

    switch (mode) {
case 0:
Serial.println("A/C Power button pressed...");
irsend.sendRaw(off, ac_ir_lenght, 38);
break;

    case 1:
Serial.println("A/C Power button pressed...");
irsend.sendRaw(on_18_, ac_ir_lenght, 38);
break;

    case 2:

    ptr = strstr(temp,"18");
if (ptr != 0){
```

```
if (f_d == 'F'){
Serial.println("A/C: 18ºC Full");
temp_18_FULL_[45] = 1516;
irsend.sendRaw(temp_18_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 18ºC Half");
temp_18_FULL_[45] = 528;
irsend.sendRaw(temp_18_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}

    ptr = strstr(temp,"19");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 19ºC Full");
temp_19_FULL_[45] = 1516;
irsend.sendRaw(temp_19_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 19ºC Half");
temp_19_FULL_[45] = 528;
irsend.sendRaw(temp_19_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}

    ptr = strstr(temp,"20");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 20ºC Full");
temp_20_FULL_[45] = 1516;
irsend.sendRaw(temp_20_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 20ºC Half");
temp_20_FULL_[45] = 528;
irsend.sendRaw(temp_20_FULL_, ac_ir_lenght, 38);
```

```
}
ptr = 0;
}

    ptr = strstr(temp,"21");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 21°C Full");
temp_21_FULL_[45] = 1516;
irsend.sendRaw(temp_21_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 21°C Half");
temp_21_FULL_[45] = 528;
irsend.sendRaw(temp_21_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}

    ptr = strstr(temp,"22");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 22°C Full");
temp_22_FULL_[45] = 1516;
irsend.sendRaw(temp_22_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 22°C Half");
temp_22_FULL_[45] = 528;
irsend.sendRaw(temp_22_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}

    ptr = strstr(temp,"23");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 23°C Full");
temp_23_FULL_[45] = 1516;
```

```
irsend.sendRaw(temp_23_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 23°C Half");
temp_23_FULL_[45] = 528;
irsend.sendRaw(temp_23_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}

    ptr = strstr(temp,"24");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 24°C Full");
temp_24_FULL_[45] = 1516;
irsend.sendRaw(temp_24_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 24°C Half");
temp_24_FULL_[45] = 528;
irsend.sendRaw(temp_24_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}

    ptr = strstr(temp,"25");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 25°C Full");
temp_25_FULL_[45] = 1516;
irsend.sendRaw(temp_25_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 25°C Half");
temp_25_FULL_[45] = 528;
irsend.sendRaw(temp_25_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}
```

```
    ptr = strstr(temp,"26");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 26°C Full");
temp_26_FULL_[45] = 1516;
irsend.sendRaw(temp_26_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 26°C Half");
temp_26_FULL_[45] = 528;
irsend.sendRaw(temp_26_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}

    ptr = strstr(temp,"27");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 27°C Full");
temp_27_FULL_[45] = 1516;
irsend.sendRaw(temp_27_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 27°C Half");
temp_27_FULL_[45] = 528;
irsend.sendRaw(temp_27_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}

    ptr = strstr(temp,"28");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 28°C Full");
temp_28_FULL_[45] = 1516;
irsend.sendRaw(temp_28_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
```

```
Serial.println("A/C: 28°C Half");
temp_28_FULL_[45] = 528;
irsend.sendRaw(temp_28_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}

    ptr = strstr(temp,"29");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 29°C Full");
temp_29_FULL_[45] = 1516;
irsend.sendRaw(temp_29_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 29°C Half");
temp_29_FULL_[45] = 528;
irsend.sendRaw(temp_29_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}

    ptr = strstr(temp,"30");
if (ptr != 0){
if (f_d == 'F'){
Serial.println("A/C: 30°C Full");
temp_30_FULL_[45] = 1516;
irsend.sendRaw(temp_30_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: 30°C Half");
temp_30_FULL_[45] = 528;
irsend.sendRaw(temp_30_FULL_, ac_ir_lenght, 38);
}
ptr = 0;
}

    break;
```

```
    case 3:
Serial.println("Blade motion pressed");
irsend.sendRaw(blades_motion_, ac_ir_lenght, 38);

    break;

    case 4:
if (f_d == 'F'){
Serial.println("A/C: Desumidify Full");
irsend.sendRaw(desumid_FULL_, ac_ir_lenght, 38);
}
else if (f_d == 'H'){
Serial.println("A/C: Desumidify Half");
irsend.sendRaw(desumid_HALF_, ac_ir_lenght, 38);
}
break;

    default:
Serial.println("Nenhum caso");
break;
}

    return;

    }
    void rf_data_decoder(char *rf_cmd){
// Função que decodifica o comando de 8 bytes recebidos no frame API // chama
as funções fan() ou ar_conditioner() com seus respectivos valores iniciais
    ptr = strstr(rf_cmd, "fan");
if (ptr != 0){
Serial.println("Controlando ventilador...");

    ptr = strstr(rf_cmd, "power");
if (ptr != 0){
fan(1);
ptr = 0;
return;
}
```

```
    ptr = strstr(rf_cmd, "slowr");
if (ptr != 0){
fan(2);
ptr = 0;
return;
}

    ptr = strstr(rf_cmd, "fastr");
if (ptr != 0){
fan(3);
ptr = 0;
return;
}

    ptr = strstr(rf_cmd, "rever");
if (ptr != 0){
fan(4);
ptr = 0;
return;
}

    else {
fan(0);
ptr = 0;
return;
}

    }

    ptr = strstr(rf_cmd, "a/c");
if (ptr != 0){
Serial.println("ar condicionado");

    char flow_d = rf_cmd[rf_data_size - 3];
char temp_d = rf_cmd[rf_data_size - 2];
char temp_u = rf_cmd[rf_data_size - 1];

    Serial.print("temperatura desejada: ");
Serial.print(temp_d);
```

```
Serial.println(temp_u);
Serial.print("potencia do ar: ");
Serial.println(flow_d);

    ptr = strstr(rf_cmd, "of___");
if (ptr != 0){
//Desliga o ar air_conditioner(0,flow_d, temp_d, temp_u);
ptr = 0;
return;
}

    ptr = strstr(rf_cmd, "onF18");
if (ptr != 0){
//Liga o ar air_conditioner(1, flow_d, temp_d, temp_u);
ptr = 0;
return;
}

    ptr = strstr(rf_cmd, "st");
if (ptr != 0){
air_conditioner(2, flow_d, temp_d, temp_u); // Troca temperatura e potencia
ptr = 0;
return;
}

    ptr = strstr(rf_cmd, "sf");
if (ptr != 0){
air_conditioner(2, flow_d, temp_d, temp_u); // Troca temperatura e potencia
ptr = 0;
return;
}

    ptr = strstr(rf_cmd, "bm");
if (ptr != 0){
air_conditioner(3, flow_d, temp_d, temp_u); // Troca temperatura e potencia
ptr = 0;
return;
}
```

```
    ptr = strstr(rf_cmd, "de");
if (ptr != 0){
air_conditioner(4, flow_d, temp_d, temp_u); // Troca temperatura e potencia
ptr = 0;
return;
}

    }

    }

    void loop() {

    if (Serial1.available() >= buf_Size){ // Se quantidade de dados do tamanho ou
maior que o buffer estiver disponivel

    //Serial.print("Bytes na porta serial: ");
Serial.println(Serial1.available());
char c = Serial1.read();

    if (c == 0x7E){

    buf[0] = c;
//Serial.print("buf[0] = ");
Serial.println(buf[0]);

    // Le cada um dos bytes e armazena em uma string
for (int i = 1; i < buf_Size + 1; i++){

    c = Serial1.read();
buf[i] = c;
//Serial.print("buf["); Serial.print(i); Serial.print("] = ");Serial.println(buf[i]);

    }

    // Pegando RF DATA do frame XBee
for (int i = 0; i < rf_data_size; i++){
rf_data[i] = buf[16 + i];
Serial.print("rf_data["); Serial.print(i); Serial.print("] = ");Serial.println(rf_-
```

```
data[i]);
}

    Serial.println(rf_data);

    // Chamar funcao que decodifica RF DATA
rf_data_decoder(rf_data);

    // reset do buffer do RF_DATA for (int i = 0; i < rf_data_size; i++){
rf_data[i] = (char)0;
//Serial.print("rf_data[i] = "); Serial.print(rf_data[i]);
}

    for (int i = 1; i < buf_Size + 1; i++){

    c = Serial1.read();
buf[i] = (char)0;

    }
Serial.println();
}
}
}
```

A.3 Aquisição de sinais IR

```
#define LEDPIN 13
//you may increase this value on Arduinos with greater than 2k SRAM
#define maxLen 800

    volatile unsigned int irBuffer[maxLen]; //stores timings - volatile because changed by ISR
volatile unsigned int x = 0; //Pointer thru irBuffer - volatile because changed by ISR

    void setup() {
Serial.begin(115200); //change BAUD rate as required
attachInterrupt(0, rxIR_Interrupt_Handler, CHANGE);//set up ISR for receiving IR signal
```

```
}
    void loop() {
// put your main code here, to run repeatedly:

    Serial.println(F("Press the button on the remote now - once only"));
delay(5000); // pause 5 secs
if (x) { //if a signal is captured
digitalWrite(LEDPIN, HIGH);//visual indicator that signal received
Serial.println();
Serial.print(F("Raw: (")); //dump raw header format - for library
Serial.print((x - 1));
Serial.print(F(") "));
detachInterrupt(0);//stop interrupts & capture until finshed here
for (int i = 1; i < x; i++) { //now dump the times
if (!(i & 0x1)) Serial.print(F(-"));
Serial.print(irBuffer[i] - irBuffer[i - 1]);
Serial.print(F(", "));
}
x = 0;
Serial.println();
Serial.println();
digitalWrite(LEDPIN, LOW);//end of visual indicator, for this time
attachInterrupt(0, rxIR_Interrupt_Handler, CHANGE);//re-enable ISR for receiving IR signal
}

    }

    void rxIR_Interrupt_Handler() {
if (x > maxLen) return; //ignore if irBuffer is already full
irBuffer[x++] = micros(); //just continually record the time-stamp of signal transitions

    }
```

Apêndice B

Códigos Aplicativo

B.0.1 Janela screen1

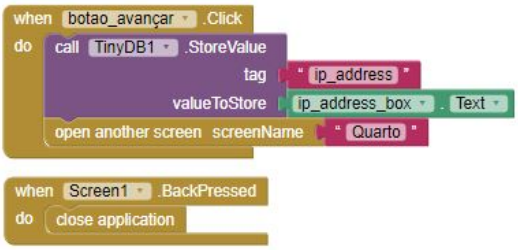

Figura B.1: Lógica de programação dos botões da janela Screen1

B.0.2 Janela Quarto

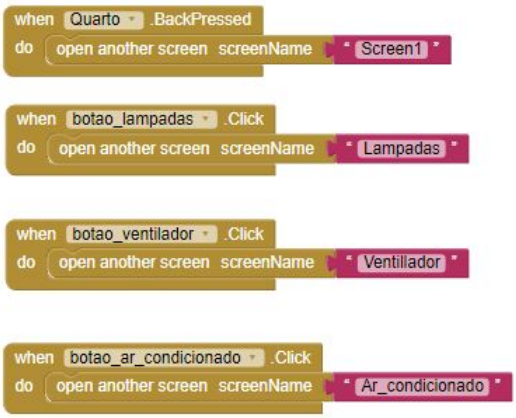

Figura B.2: Lógica de programação dos botões da janela Quarto

B.0.3 Janela Lampadas

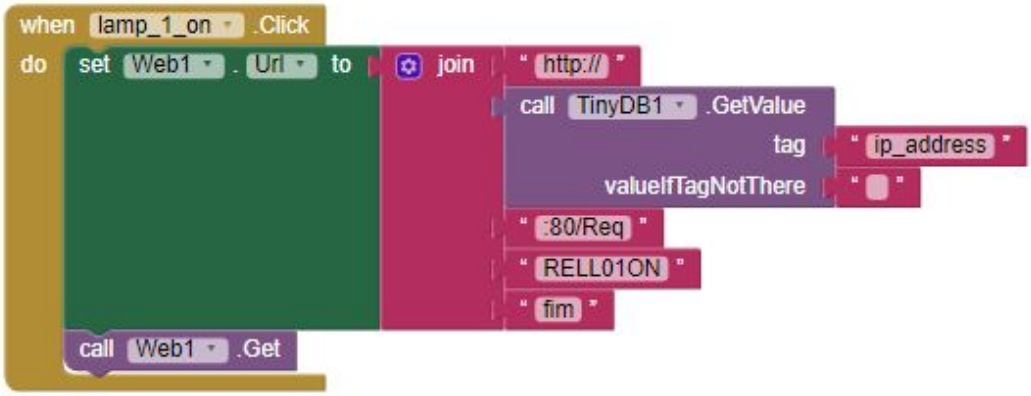

Figura B.3: Lógica de programação do botão Acender da lâmpada 1

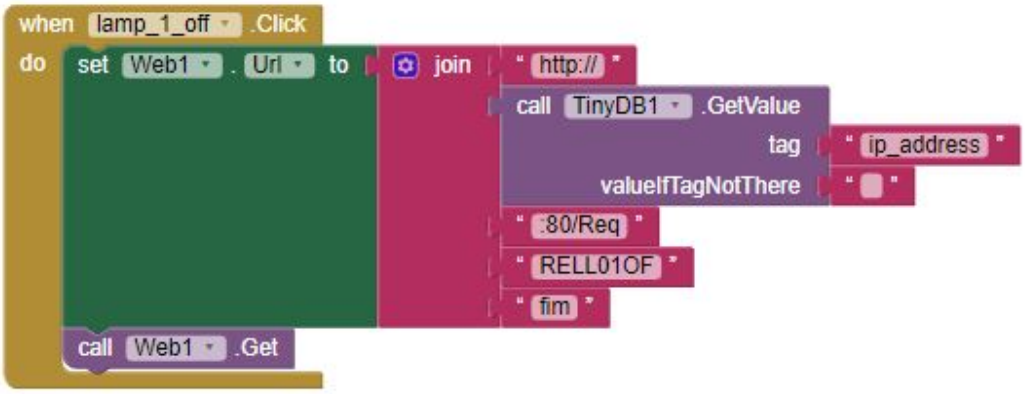

Figura B.4: Lógica de programação do botão Apagar da lâmpada 1

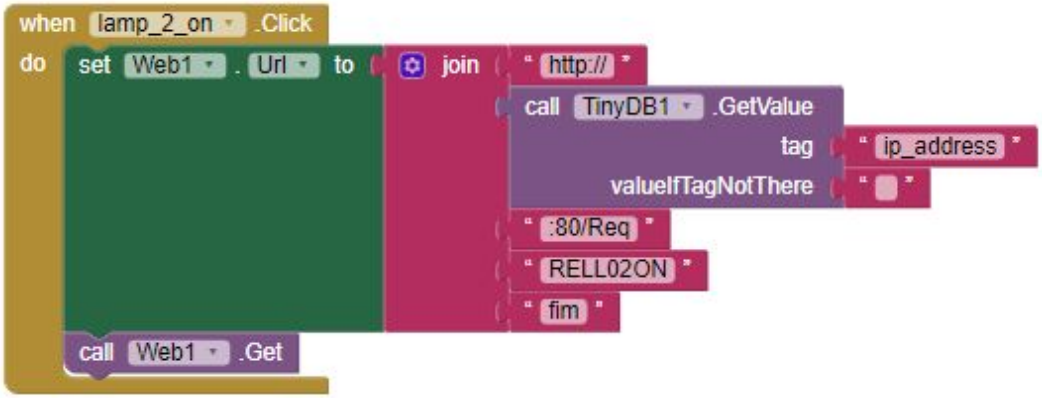

Figura B.5: Lógica de programação do botão Acender da lâmpada 2

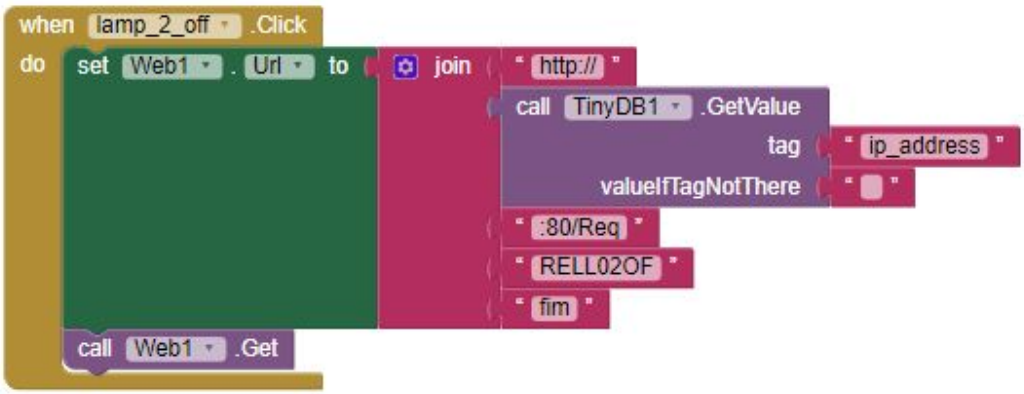

Figura B.6: Lógica de programação do botão Apagar da lâmpada 2

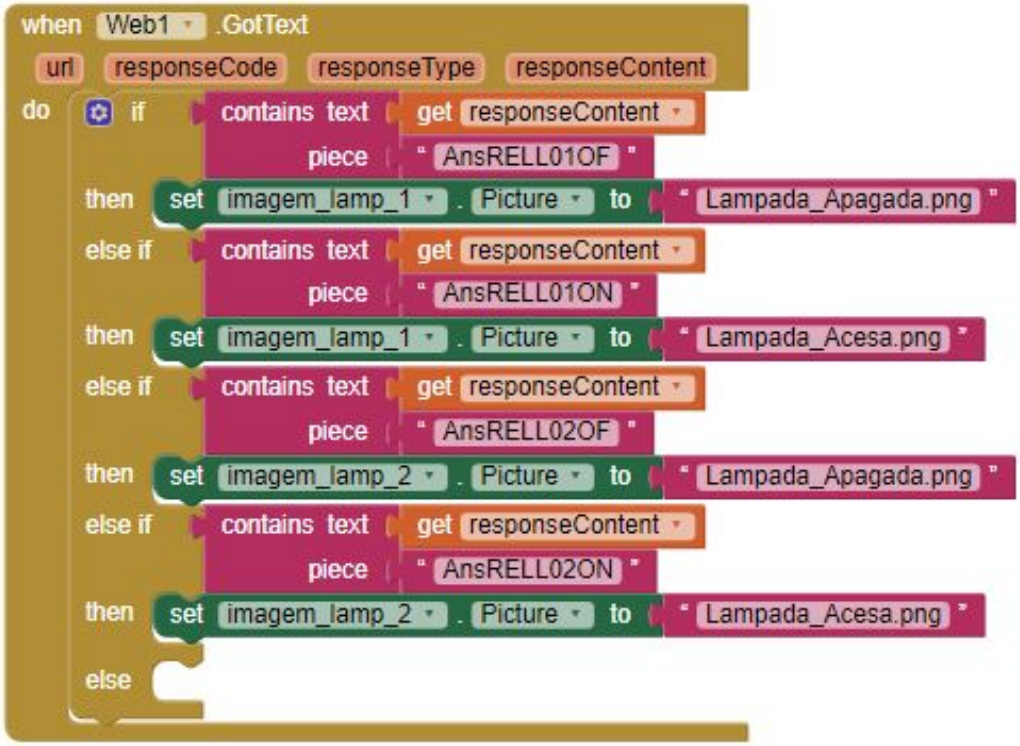

Figura B.7: Lógica de programação de recepção da resposta html

Figura B.8: Lógica de programação do botão voltar do dispositivo

B.0.4 Janela Ventilador

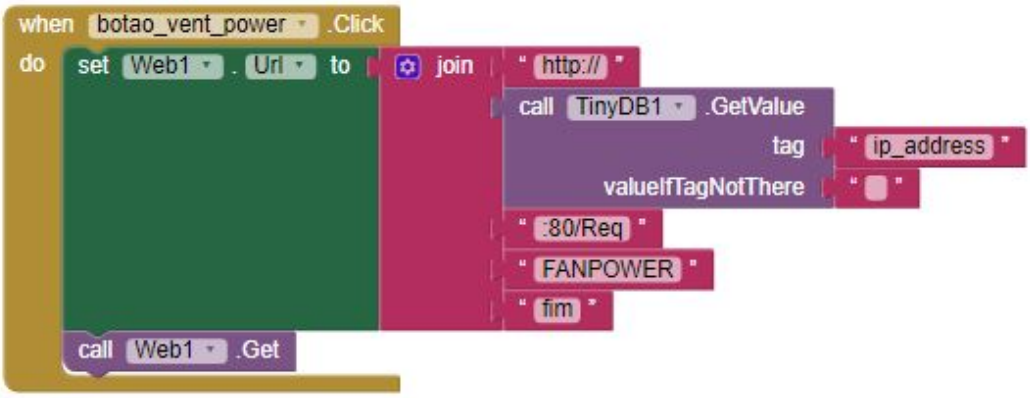

Figura B.9: Lógica de programação do botão power do ventilador

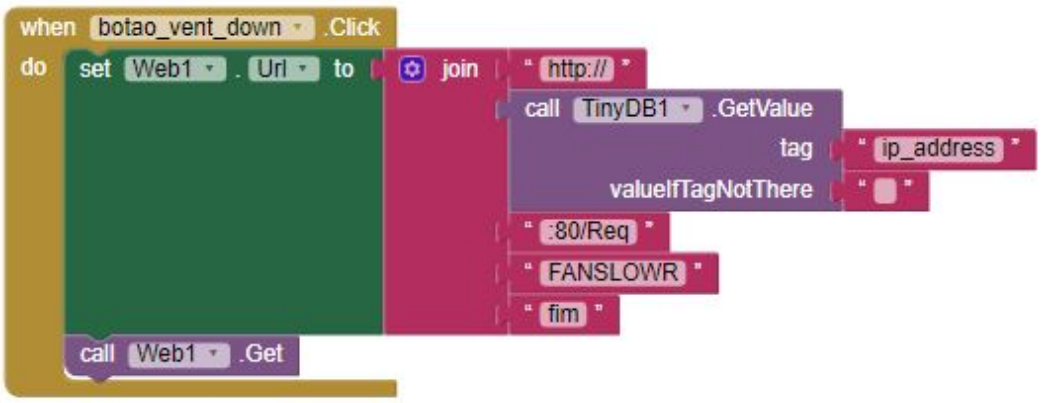

Figura B.10: Lógica de programação do botão slow dwon do ventilador

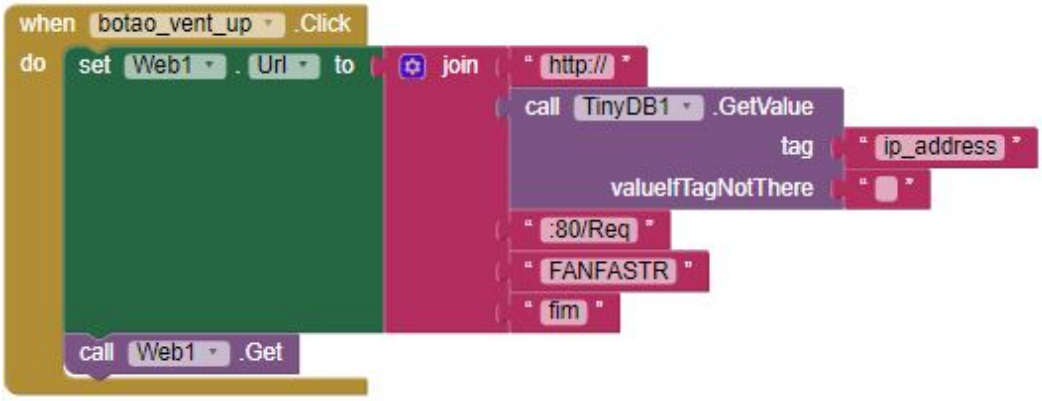

Figura B.11: Lógica de programação do botão speed up do ventilador

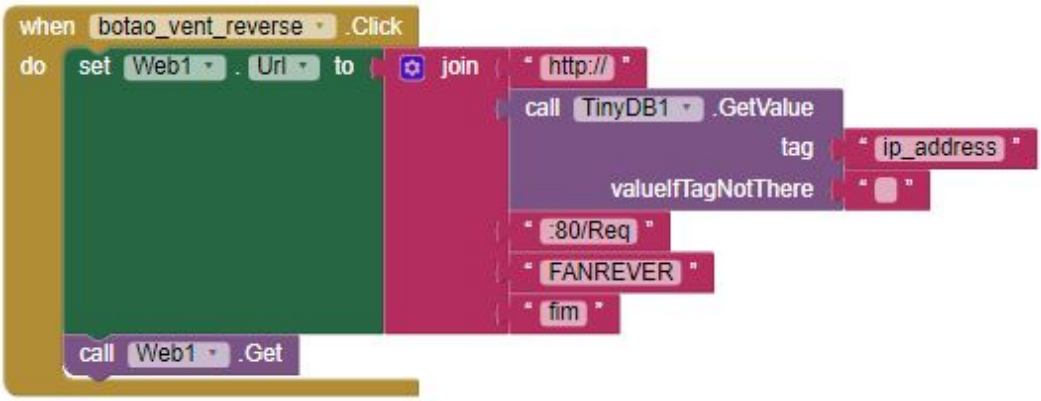

Figura B.12: Lógica de programação do botão reverse do ventilador

Figura B.13: Lógica de programação do botão voltar do dispositivo

B.0.5 Janela Ar condicionado

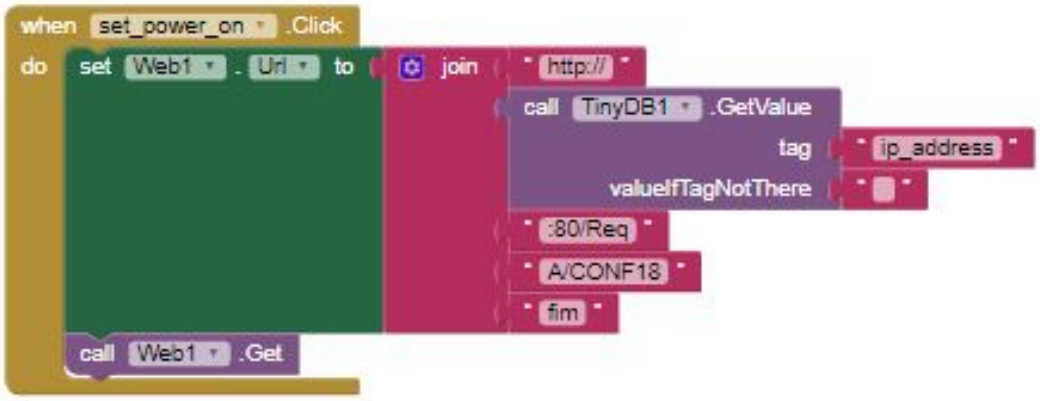

Figura B.14: Lógica de programação do botão On do ar condicionado

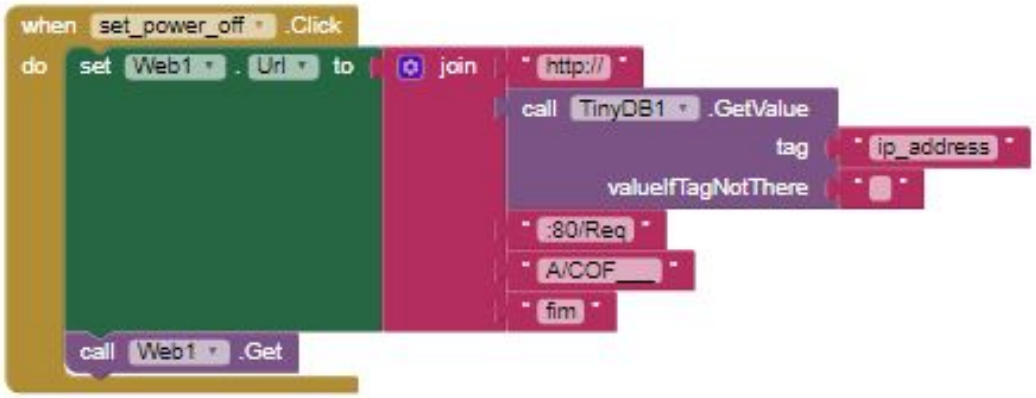

Figura B.15: Lógica de programação do botão Off do ar condicionado

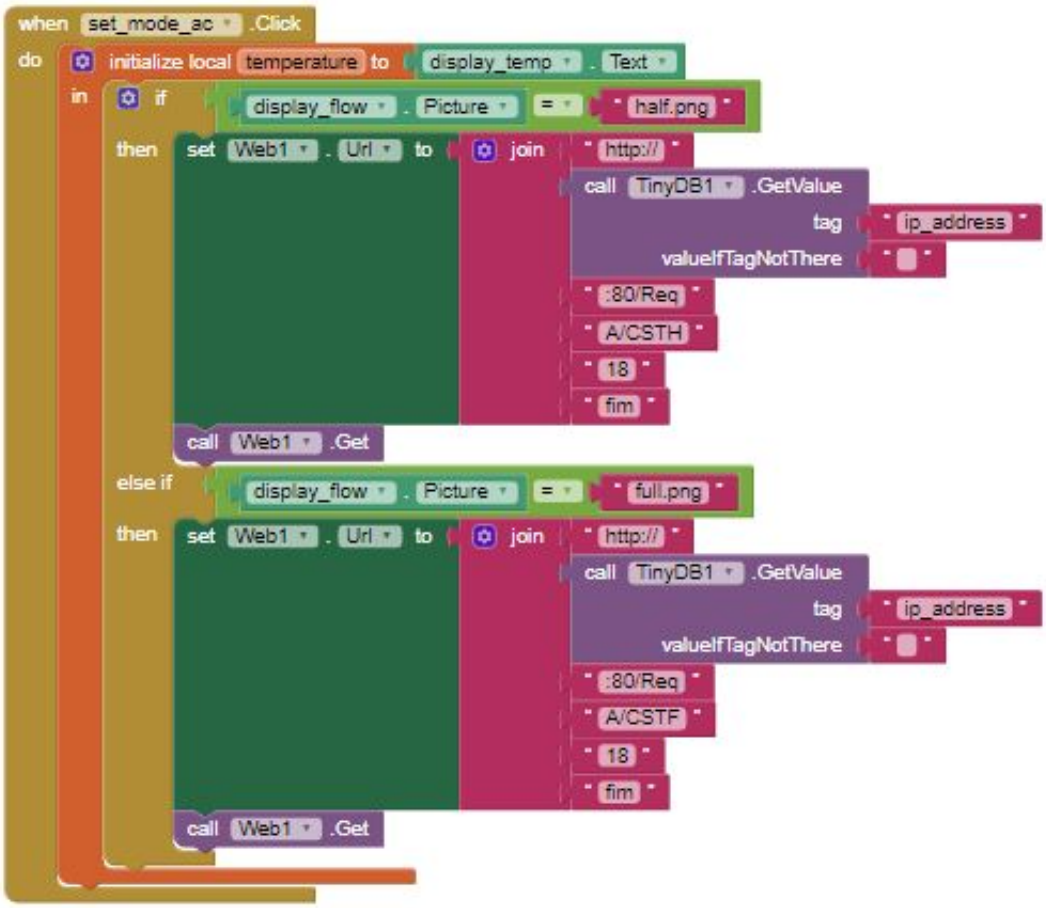

Figura B.16: Lógica de programação do botão AC do ar condicionado

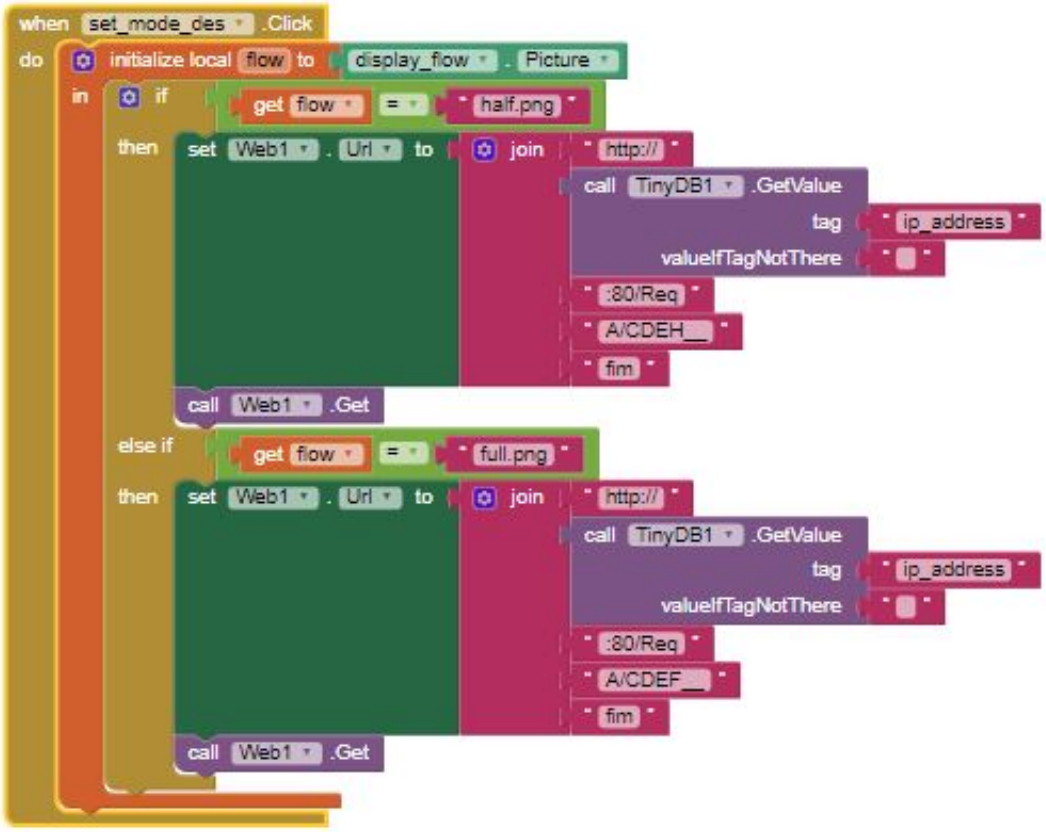

Figura B.17: Lógica de programação do botão DES do ar condicionado

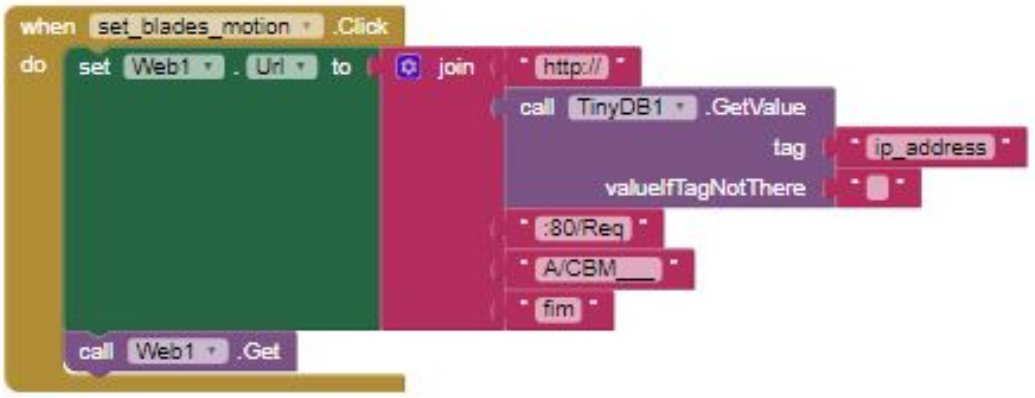

Figura B.18: Lógica de programação do botão Move/Stop do ar condicionado

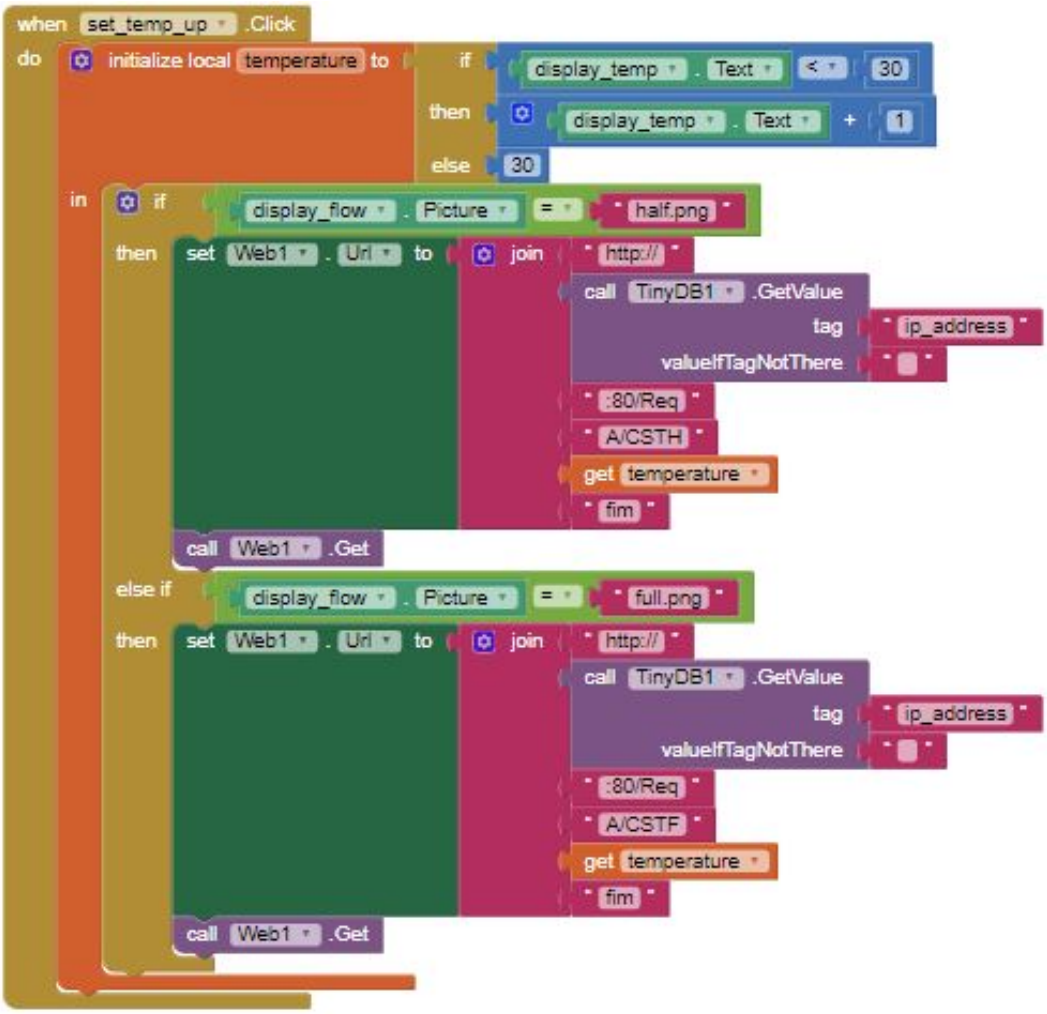

Figura B.19: Lógica de programação do botão de aumentar temperatura do ar condicionado

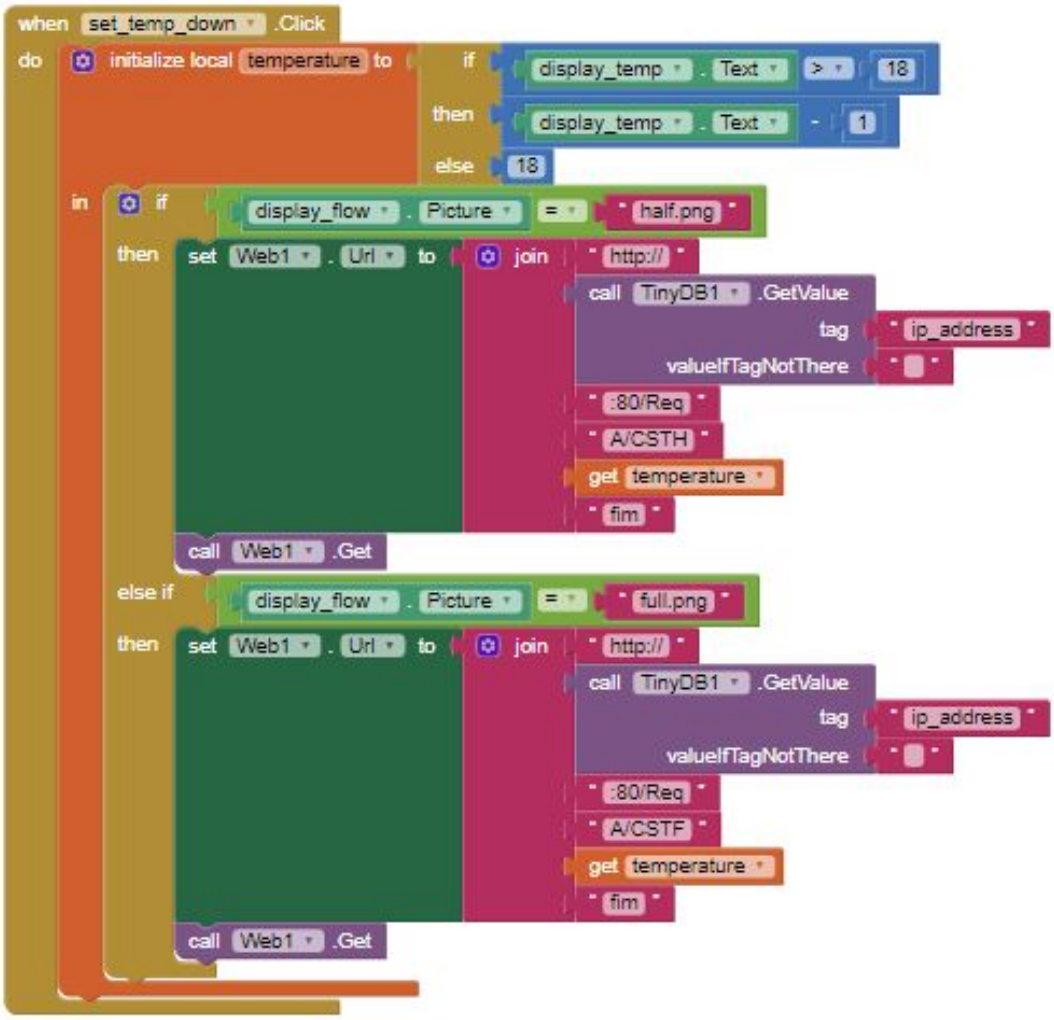

Figura B.20: Lógica de programação do botão de diminuir a temperatura do ar condicionado

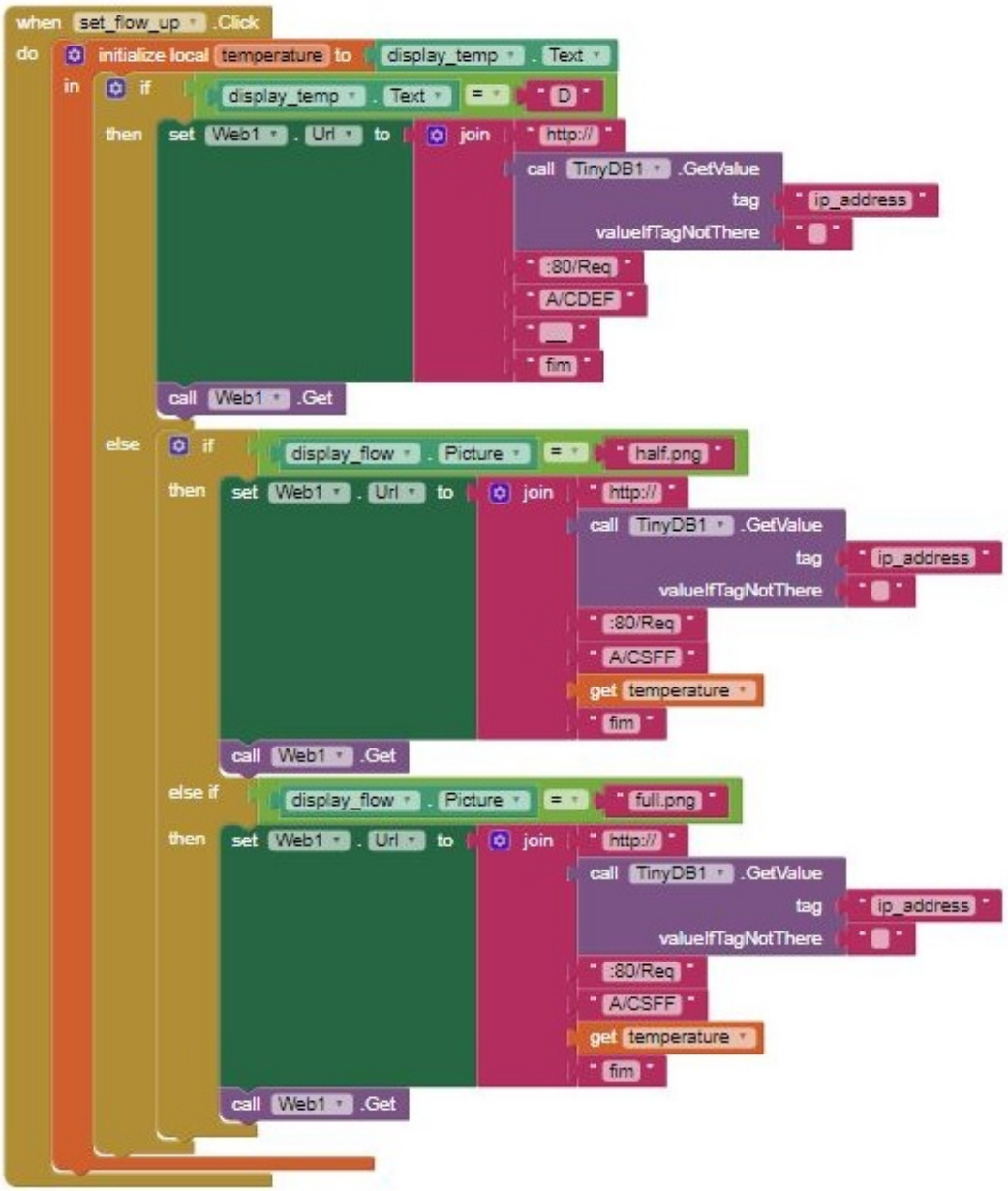

Figura B.21: Lógica de programação do botão de aumentar vazão do ar condicionado

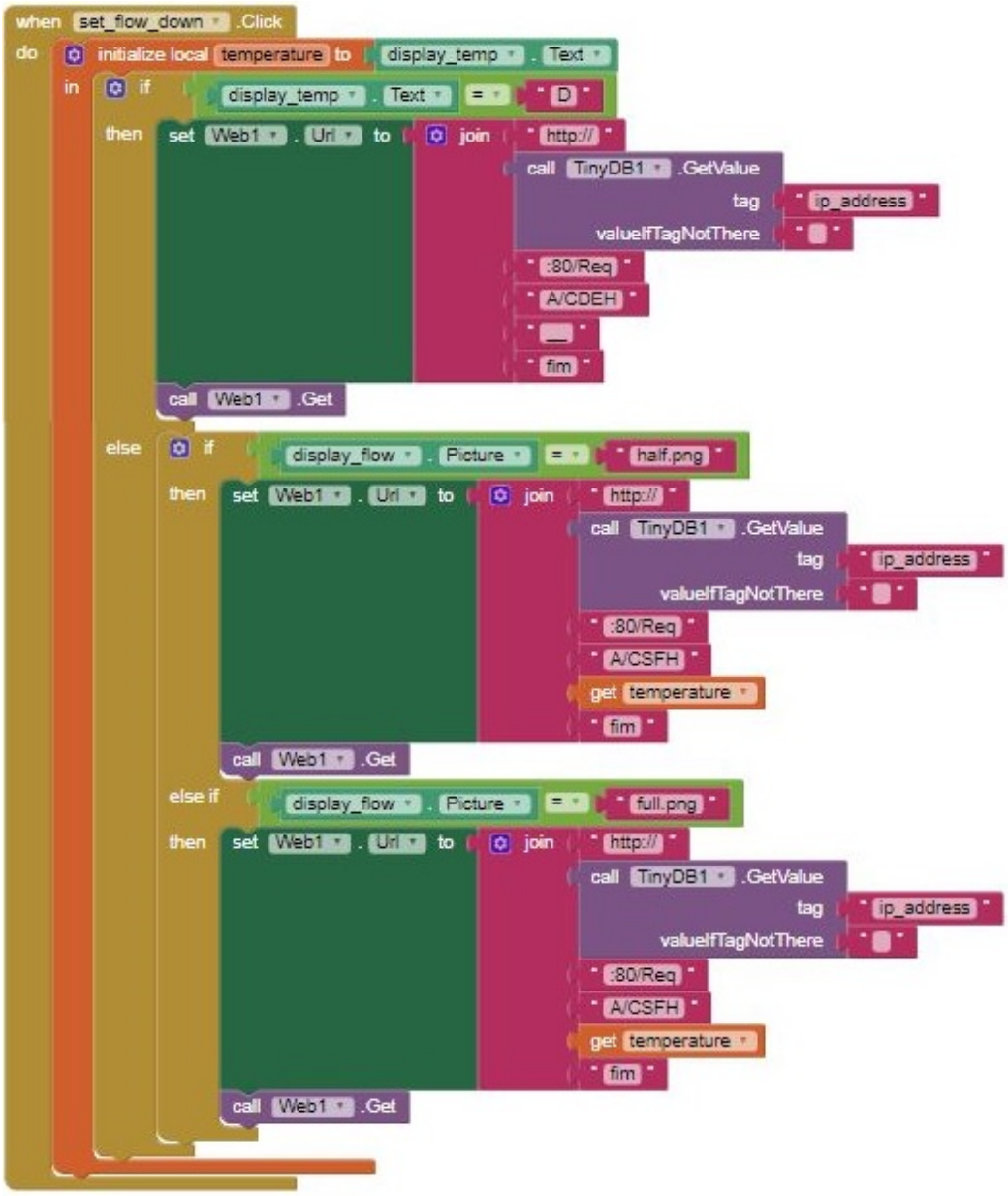

Figura B.22: Lógica de programação do botão de diminuir vazão do ar condicionado


when Web1 .GotText
url responseCode responseType responseContent
do if contains text get responseContent piece " AnsA/CST_18 "
then set display_temp . Text to " 18 "
else if contains text get responseContent piece " AnsA/CST_19 "
then set display_temp . Text to " 19 "
else if contains text get responseContent piece " AnsA/CST_20 "
then set display_temp . Text to " 20 "
else if contains text get responseContent piece " AnsA/CST_21 "
then set display_temp . Text to " 21 "
else if contains text get responseContent piece " AnsA/CST_22 "
then set display_temp . Text to " 22 "
else if contains text get responseContent piece " AnsA/CST_23 "
then set display_temp . Text to " 23 "
else if contains text get responseContent piece " AnsA/CST_24 "
then set display_temp . Text to " 24 "
else if contains text get responseContent piece " AnsA/CST_25 "
then set display_temp . Text to " 25 "
else if contains text get responseContent piece " AnsA/CST_26 "
then set display_temp . Text to " 26 "
else if contains text get responseContent piece " AnsA/CST_27 "
then set display_temp . Text to " 27 "
else if contains text get responseContent piece " AnsA/CST_28 "
then set display_temp . Text to " 28 "
else if contains text get responseContent piece " AnsA/CST_29 "
then set display_temp . Text to " 29 "
else if contains text get responseContent piece " AnsA/CST_30 "
then set display_temp . Text to " 30 "
else if contains text get responseContent piece " AnsA/CSFFTT "
then set display_flow . Picture to " full.png "
else if contains text get responseContent piece " AnsA/CSFHTT "
then set display_flow . Picture to " half.png "
else if contains text get responseContent piece " AnsA/CDEHTT "
then set display_temp . Text to " D "
set display_flow . Picture to " half.png "
else if contains text get responseContent piece " AnsA/CDEFTT "
then set display_temp . Text to " D "
set display_flow . Picture to " full.png "
else if contains text get responseContent piece " AnsA/CONF18 "
then set display_temp . Text to " 18 "

Figura B.24: Lógica de programação do botão voltar do dispositivo