UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADEMICO DE INFORMÁTICA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MARLON BRUNO VENDRAMETTO

AUTOMAÇÃO RESIDENCIAL UTILIZANDO MICROCONTROLADOR ARDUINO E APLICATIVO MÓVEL

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2018

MARLON BRUNO VENDRAMETTO

AUTOMAÇÃO RESIDENCIAL UTILIZANDO MICROCONTROLADOR ARDUINO E APLICATIVO MÓVEL

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Augusto Foronda

PONTA GROSSA

2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

AUTOMAÇÃO RESIDENCIAL UTILIZANDO MICROCONTROLADOR ARDUINO E APLICATIVO MÓVEL

por

MARLON BRUNO VENDRAMETTO

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 23 de outubro de 2018 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Augusto Foronda
Orientador

_____
Prof. Dr. Diego Roberto Antunes
Membro titular

_____
Prof. Dr. Richard Duarte Ribeiro
Membro titular

_____
Prof.ª Dra. Helyane Bronoski Borges
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof. Dr. André Pinz Borges
Coordenador do curso

- A Folha de Aprovação assinada encontra-se arquivada na Secretaria Acadêmica -

RESUMO

VENDRAMETTO, Marlon Bruno. **Automação residencial utilizando microcontrolador Arduino e aplicativo móvel**. 2018. 57 f. Trabalho de Conclusão do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

Este trabalho tem o objetivo de aplicar o conceito de Internet das Coisas em uma residência, permitindo o controle dos recursos através de um aplicativo para celular. Primeiramente explica o funcionamento do módulo ESP12E e a comunicação com os sensores. O módulo possui um microcontrolador e a sua forma de funcionamento assemelha-se à dos dispositivos Arduino, porém com um recurso a mais que é a capacidade de troca de informações com uma rede sem fio. Essa comunicação ocorre quando o dispositivo móvel envia requisições para o servidor, que transmite ao ESP12E, e por meio do código armazenado em linguagem C na sua memória interna, analisa as instruções dentro das condições estabelecidas e envia sinais elétricos para os sensores ou atuadores que estão conectados e auxiliam no controle. Primeiramente, implementa-se o controle de um LED acendendo e apagando, que representa o controle de uma luz da residência. Na sequência, um sensor de ultrassom e uma sirene que emite um alerta sonoro quando é verificada a presença de um intruso e assim representa um sistema de alarme. Por fim, um atuador servo motor que realiza a abertura e o fechamento de uma porta. Todos estes recursos podem ser manipulados a partir do toque na tela do *smartphone,* que foi elaborado em uma linguagem multiplataforma, baseado no *framework* Ionic que possibilita desenvolvimento de apenas um código capaz de executar paralelamente em dois sistemas operacionais diferentes, Android e iOS. Esta pesquisa permite ao leitor aplicar na prática o conceito de automação residencial e ainda a expansão do mesmo, aperfeiçoando o controle de dados para garantir a economia sustentabilidade e de uma família.

Palavras-chave: Automação residencial. ESP12E. *Framework* Ionic.

ABSTRACT

VENDRAMETTO, Marlon Bruno. **Home automation using microcontroller Arduino and mobile application**. 2018. 57 p. Work of Conclusion Course - Graduation in Analysis and Systems Development - Federal Technology University - Paraná. Ponta Grossa, 2018.

This work aims to apply the concept of Internet of Things in a residence, allowing the control of the resources through a mobile application. Firstly, explain the functions of ESP12E module and communication with sensors. The module has a microcontroller and its operation seems with the Arduino tools, but with the ability to exchange information with a wireless network. This communication occurs when the mobile application send requests to the server, that transmits to ESP12E, and through the armazenated code in C language in your memory, analise the instructions inside of conditions fixed and send eletric signals to the sensors or actuators that be conected and help to control. First, implementing control of an LED that turns on or off and represents control of a residence light. Thereafter, an ultrasound sensor and a siren that emits an audible alert when an intruder's presence is verified and thus represents an alarm system. Finally, a servo motor actuator that opens and closes a door. All features can be manipulated by touch in the smartphone screen, which is built in a cross-platform language, based on the Ionic framework that enables the development of one code capable of running in parallel on two different systems, Android and iOS. This research allows the reader to apply the concept of residential automation and maintain the expansion of it, improving data control to ensure economic and family sustainability.

Keywords: Home automation. ESP12E. Ionic *Framework*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	*Application Programming Interface*
CSS	*Cascading Style Sheets*
HTML	*HypertText Markup Language*
HTTP	*HypertText Transfer Protocol*
IDE	*Integrated Development Environment*
IOT	*Internet of Things*
LED	*Light Emitting Diode*
PC	*Personal Computer*
SDK	*Software Development Kit*
SO	Sistema Operacional
USB	*Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO

A tecnologia é constantemente explorada e redesenhada de uma maneira que melhore a qualidade de vida do ser humano. Quanto mais a população mundial cresce, mais se torna necessário que os recursos tecnológicos sejam aprimorados, nos trazendo vidas mais saudáveis, gratificantes e confortáveis (EVANS, 2011).

Nos dias de hoje, ao entrar-se no assunto de tecnologia, é necessário falar da Internet, pois grande parte dos dispositivos, como TVs, *Laptops*, automóveis, *smartphones*, consoles de jogos, já possuem este recurso embutido, evidenciando o conceito de Internet das coisas (IoT – *Internet of Things*) (SANTOS et al, 2016).

A IoT se trata de uma extensão do conceito de *internet*, no qual a mesma é inserida nos objetos e somada aos sensores ou atuadores, permite que eles se comuniquem e interajam com o ambiente no qual estão inseridos. Através destes objetos é possível perceber o meio externo, manipula-lo e ainda garantir a interação de um objeto com outro e também com as pessoas (SANTOS et al, 2016).

Outro assunto importante a ser abordado quando se trata de tecnologia, é a evolução dos *smartphones* e a utilização deles no dia-a-dia do brasileiro como revela a pesquisa realizada em 2017 pela empresa Deloitte sobre o celular no dia a dia do brasileiro. No ano de 2017, o dispositivo eletrônico mais utilizado pelos brasileiros foi o *smartphone*, com um aumento de utilização de 7% em relação ao ano anterior. Outro dado importante foi o hábito dos entrevistados em fazer uso do pacote de dados até ultrapassar o valor contratado, “8 em cada 10 entrevistados estouram o pacote de dados”, afirma a pesquisa. Isso indica que os brasileiros permanecem conectados boa parte do dia através de redes móveis e se for analisada com mais atenção, a pesquisa não leva em consideração o uso de redes *wireless*, que evidenciaria ainda mais a mudança de hábito do brasileiro (DELOITTE, 2017, p. 5).

Estas duas vertentes tecnológicas, os *smartphones* sendo utilizados para realizar as mais diversas atividades e, os objetos inteligentes que interagem com o ambiente, precisam integrar-se e assim somar seus benefícios para aprimorar a qualidade de vida dos indivíduos. O dispositivo móvel funciona como um controle remoto que através da Internet se comunica com os objetos e permite o controle dos mesmo em qualquer lugar, necessitando apenas de acesso à *web*.

Assim, a proposta deste trabalho consiste em desenvolver um aplicativo móvel que manipule através de microcontroladores os recursos de uma residência, tais como: o acionamento e desligamento de luz, sensor ultrassônico e sonoro, e também controle do mecanismo de abertura e fechamento de uma porta residencial.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo do trabalho é desenvolver um aplicativo móvel para automação dos recursos de uma residência com o microcontrolador ESP12E.

1.1.2 Objetivos Específicos

- Demonstrar o funcionamento do ESP12E e a comunicação com os sensores;
- Desenvolver um código para o funcionamento do LED, sensor ultrassom, alarme sonoro e servo motor;
- Desenvolver um aplicativo móvel para controlar os sensores e atuadores;
- Realizar a integração entre o ESP12E e a aplicação móvel;
- Criar um protótipo da residência posicionando os sensores.

1.2 JUSTIFICATIVA

A automação residencial pode trazer diversos benefícios a quem a utiliza. Através de um aplicativo móvel pode-se obter maior praticidade na execução das tarefas diárias, como por exemplo apagar uma luz, ativar o alarme, abrir uma porta, dentre tantas outras possibilidades.

O uso do aplicativo móvel também permite o controle de uma casa em tempo real, utilizando câmeras e outros dispositivos de segurança pode-se monitorar

eventos ocorridos durante o dia. Por exemplo, quando o alarme dispara e o proprietário está no trabalho, o mesmo pode então consultar as câmeras imediatamente, ver o motivo, e tomar as ações necessárias.

Este trabalho busca uma solução para estes problemas e também para a falta de tempo que o mundo globalizado nos trouxe. Apresenta uma amostra prática de como a IoT pode facilitar nossas vidas e centraliza o controle de uma residência em um aplicativo que pode ser acessado remotamente, gerenciando os recursos em tempo real.

2 REFERENCIAL TEÓRICO

Este capítulo inicia-se abordando a Internet das Coisas e os recursos lógicos e físicos que compõem o funcionamento da mesma, em específico o Arduino, seus módulos e também sobre os sensores e atuadores, recursos auxiliares ao microcontrolador que realizam a troca e captura de informações com o ambiente externo.

Por fim, abordam-se as principais tecnologias disponíveis para dispositivos móveis, em específico a linguagem híbrida e um exemplo de *framework* que utiliza este padrão: o *Ionic Framework*.

2.1 INTERNET DAS COISAS

O computador faz parte do cotidiano dos seres humanos desde que se tornou acessível. De lá para cá, seu formato físico foi reduzido e sua capacidade de processamento tornou-se mais eficiente, possibilitando a criação dos *tablets* e *smartphones*. O século XX é marcado por uma forte evolução das tecnologias referentes a comunicação, abriram-se várias possibilidades para a troca de informação. A televisão, a telecomunicação e a informática, passaram a integrar-se. Isso ocasionou uma revolução na forma com que as mídias utilizavam seus recursos e também no modo de vida dos seres humanos, tornando tudo digital (FONSECA FILHO, 2007).

Com a integração dos recursos, os indivíduos se tornaram mais familiarizados com a tecnologia e o mundo virtual. Os celulares inteligentes passaram a ser utilizados para desempenhar diversas atividades cotidianas, poupando tempo e facilitando a comunicação. Isso foi possível devido a introdução da Internet nos celulares e de acordo com a autora SANTAELLA (2008), estar conectado à rede não diz respeito somente à interação entre dados e pessoas, mas sim de mais um terceiro elemento: as coisas (SANTAELLA, 2008).

Não somente os celulares como diversos outros dispositivos eletroeletrônicos aprimoraram-se recebendo Internet. Além de executarem suas funções básicas também estão mais inteligentes e são capazes de tomar algumas

decisões básicas sozinhos. Com o auxílio de sensores, pode-se compreender alterações no ambiente externo e os programar para executarem alguma ação a partir disso, se tornando dinâmicos e comunicantes (SANTAELLA, 2008).

Em um exemplo prático, recentemente (2016) a Revista Exame publicou um artigo sobre a OMO (marca de detergentes em pó). A marca está desenvolvendo um dispositivo com o nome de “Peggy”, que funciona de forma semelhante a um grampo de roupas normal. Porém, ele possui sensores de luz, umidade e vento integrados que percebem quando há possibilidade de chuva e avisa ao usuário através de um aplicativo para celular. Pode ainda indicar qual o melhor horário do dia para lavar as roupas (REVISTA EXAME, 2016).

Pode-se citar também a utilização dos sensores embutidos para identificar ações do usuário ou adaptar suas configurações às condições do meio externo. Através dos sensores é possível identificar a comunicação corporal do indivíduo, mesmo sem ele ter pressionado uma tecla ou deslizado seu dedo sobre a tela, permitindo que o brilho da tela seja ajustado automaticamente quando perceber ambientes com pouca luz, por exemplo (ERTHAL, 2007).

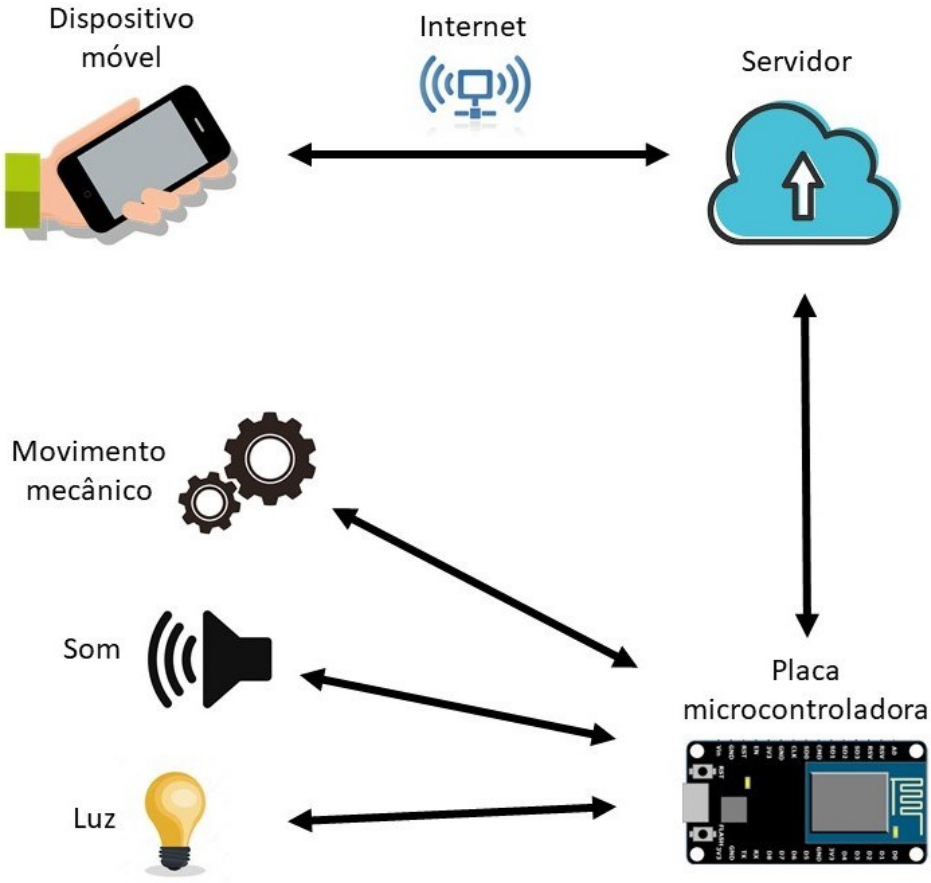
Conclui-se que a Internet das Coisas “não é somente ligar as “coisas” pela internet, mas também torná-las inteligentes, capazes de coletar e processar informações do ambiente ou das redes às quais estão conectadas” (OLIVEIRA, 2017, p. 17).

Para que a IoT possa funcionar e o gerenciamento dos recursos seja possível, precisa-se primeiramente de uma placa com um microcontrolador. Nela será armazenado o código para controle dos sensores ou atuadores e também a configuração da forma de comunicação com a internet, para que a interação com outros dispositivos se realize. Após isso, tem-se a necessidade de criar uma interface para que o usuário possa realizar ações e assim alterar o estado dos componentes (SOUZA, 2016).

Após configurar-se os componentes, um exemplo prático de como o funcionamento pode ocorrer se dá pela seguinte forma: o usuário seleciona a ação que deseja executar pelo aplicativo no dispositivo móvel, que por sua vez emite um sinal (via *web*) para o servidor criado pela placa microcontroladora. O servidor recebe o comando e envia para a placa que faz o gerenciamento dos demais elementos convertendo para os sensores ou atuadores o sinal elétrico em luminoso,

sonoro ou mecânico. Existe também o caminho inverso, no qual um dos sensores envia informações do seu estado atual para a placa para ser apresentado na interface. Na Figura 1, evidencia-se a interação destes elementos (SOUZA, 2016).

Figura 1 – Topologia de um sistema de automação residencial



Fonte: Adaptado de Souza (2016).

Nos próximos tópicos abordam-se os principais elementos que compõem a figura 1 e que estão presentes na aplicação prática deste trabalho.

2.2 PLACAS MICROCONTROLADORAS

Ao comparar as placas de circuitos eletrônicos que compunham os primeiros computadores com as de hoje em dia, nota-se uma grande transformação e isso se deve ao avanço da tecnologia, onde o tamanho foi reduzido e a capacidade de

processamento aumentada (adotando nomenclaturas como Micro, Nano), para que fossem acoplados não somente aos notebooks e celulares, mas também em carros, eletrodomésticos, brinquedos infantis, etc. Hoje são conhecidos como microcontroladores (MONK, 2013).

Como afirma Monk (2013), “um microcontrolador é um pequeno computador dentro de um chip”. Sua estrutura é muito parecida como a de um PC comum, possui um processador, uma memória RAM, uma memória *flash* e, além disso, portas de entrada e saída que servem para realizar a conexão com os demais componentes (MONK, 2013, p. 6).

Um microcontrolador consegue realizar toda a parte de gerência de outros dispositivos que estejam ligados a ele. Através de sensores e atuadores, podem-se captar informações do ambiente externo transformando-as em dados ou traduzindo comandos enviados de um ponto externo indicando ações ou movimentos mecânicos, respectivamente (MONK, 2013).

Um exemplo de sistema que se baseia nessa tecnologia é o Arduino, que é muito utilizado no mundo todo devido seu baixo custo. A patente de fabricação é livre, qualquer empresa que deseje fabricar uma cópia desta placa e vender pode fazer isso livremente. A linguagem utilizada é *open source* (C e C++) e possui uma comunidade *online* ativa que contribui nos diversos fóruns sobre o assunto (MONK, 2013).

Criada com o intuito de auxiliar no ensino de estudantes e mais tarde se tornando comercial, o Arduino se popularizou pelos motivos já citados acima. Possui alguns modelos classificados como famílias e apresenta um conjunto de módulos que podem funcionar de maneira complementar ao seu funcionamento ou de forma totalmente independente (MONK, 2013).

O primeiro passo para se iniciar um projeto com esse equipamento é acessar a IDE do Arduino, que pode ser baixada gratuitamente no site do fabricante. Na plataforma de programação, deve-se então criar um código em uma das linguagens aceitas, indicando as ações a serem executadas através de condições lógicas. Após isso, deve-se plugar o cabo USB no computador e a outra ponta no dispositivo para então carregar a instrução (MONK, 2013).

A partir deste momento o dispositivo armazena o código e já está pronto para ser utilizado, mesmo se desconectar o USB do computador. Isso ocorre porque

a instrução fica armazenada dentro da memória da placa e pode ser acessada pelo CPU a qualquer momento (MONK, 2013).

As conexões digitais, são enumeradas e podem ser usadas como entradas e saídas. Quando utilizados como saída, trabalham com tensões elétricas, alternando entre 0V (desligadas) e 5V (ligadas) (MONK, 2013).

O microcontrolador da placa é a parte mais importante do dispositivo como um todo, pois é responsável por controlar todos os outros recursos. O microcontrolador acessa a memória para consultar as instruções e as executa, além disso, também pode alterar dados contidos na memória ou, então, a tensão de uma saída digital (MONK, 2013).

Na parte inferior da placa, localiza-se o grupo de conexões de alimentação elétrica, onde o primeiro botão é o Reset, que tem a função de reiniciar o sistema. Ao se pressionar o botão de Reset, a execução do *sketch* (algoritmo) é reposicionada para o início da instrução (MONK, 2013).

Ao lado das conexões, encontram-se as entradas analógicas, de A0 a A5, que são utilizados para identificar a tensão com que cada pino está trabalhando (MONK, 2013).

Através de suas conexões digitais e entradas analógicas, pode-se conectar outros dispositivos para realizar as ações de acordo com o objetivo do projeto, como por exemplo, os sensores ou atuadores, como se verifica nos próximos capítulos.

2.2.1 ESP8266

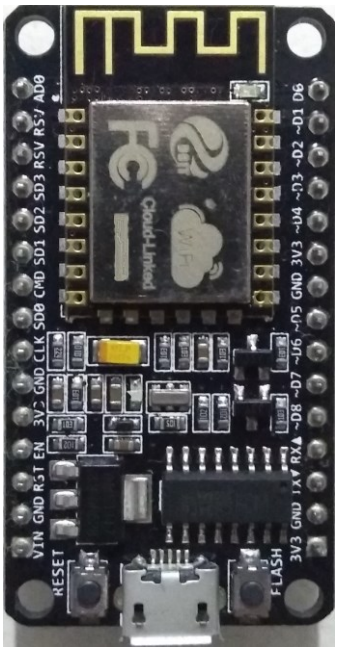
Apesar de uma placa Arduino possuir muitos recursos, verifica-se que ela é genérica. Por exemplo, ela não consegue funcionar como um servidor *web*, nem controlar um motor, um dispositivo USB ou um relé, devido a não ter em seu dispositivo um módulo específico para realizar essas atividades. Isso se explica pelo fato de que embutir vários recursos em uma placa iria encarecer sua fabricação, além de obrigar alguns usuários da plataforma a pagarem por recursos que não irão utilizar nos seus projetos (MONK, 2013).

Pensando nisso, foram criados os *Shields* ou Módulos para controle de recursos específicos que podem ser plugados ao Arduino, ou ainda tem seu

funcionamento independente dele. Com estes módulos pode-se agregar funções adicionais ao Arduino ou utilizar apenas suas funções em um projeto específico. Os módulos eliminam a necessidade de soldar fios para interligar outras placas ao Arduino (MONK, 2013).

Um exemplo de um desses módulos é o ESP12E, conforme a Figura 2.

Figura 2 – Módulo ESP12E



Fonte: Autoria própria

Este *chip* funciona de forma muito semelhante a um Arduino convencional, com os mesmos recursos de portas digitais (D0 a D8, SD2 e SD3, também chamadas de GPIO), instrução armazenada, botão *reset* e *flash*. Porém, sua principal característica e vantagem sobre a outra reside no fato de que ela possui o recurso de conexão *wireless*, possibilitando a comunicação com um servidor.

2.3 SENSORES E ATUADORES

Como citado anteriormente, o Arduino e seus módulos podem fazer conexões com outras plataformas para obter dados do ambiente externo ou exercer

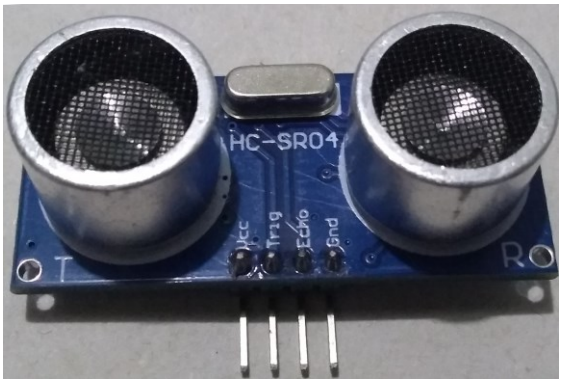
alguma atividade a partir de sua instrução. Para isso, é necessário o uso de Sensores ou Atuadores.

2.3.1 Sensores

Um sensor é um dispositivo sensível a alterações do ambiente, consegue captar as variações de luz, térmica, som e transforma em números, grandezas físicas tais como: temperatura, pressão, velocidade, corrente, aceleração, posição e outros (WENDLING, 2010).

Um exemplo de dispositivo bastante utilizado é o sensor ultrassônico HC-SR04, como apresentado na Figura 3.

Figura 3 – Sensor ultrassônico HC-SR04



Fonte: Autoria própria

Este sensor possui quatro pinos, sendo o primeiro VCC, por onde a placa recebe a energia para funcionamento. Em seguida tem-se o TRIG (de *trigger*), que realiza o envio de um sinal ultrassônico, para que então o ECHO capture o retorno do sinal e calcule a distância entre o objeto e o sensor. Por fim apresenta o GND, com a função de aterramento.

2.3.2 Atuadores

Atuadores são componentes que recebem um estímulo que pode ser elétrico, hidráulico ou pneumático através de um microcontrolador e têm a capacidade de transforma-lo em movimentos mecânicos, sonoros ou luminosos (DUTRA; ROMANO, 2016).

O termo LED (*Light Emitting Diode*) é utilizado para identificar um diodo emissor de luz. Normalmente é utilizado em projetos para representar alterações de estado de objetos ou responder a eventos. Possui dois polos, como pode-se verificar na Figura 4, o maior sempre será o positivo e o menor, negativo (ALVES et al, 2012).

Figura 4 – Led

Fonte: Autoria própria

Outro exemplo de atuador é o *Buzzer*, conforme verifica-se na Figura 5, que emite sons a partir de um estímulo elétrico e é utilizado para representar um alarme, despertador ou qualquer outro recurso que utilize sinal sonoro. Seus pinos funcionam de forma idêntica ao de um LED (positivo e negativo).

Figura 5 – Buzzer DC 5V

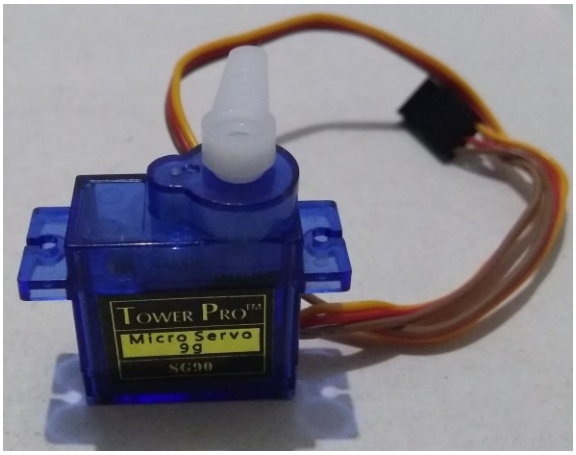
Fonte: Autoria própria

Ainda sobre os atuadores, pode-se citar o micro servo motor que transforma um estímulo elétrico em mecânico através do recebimento de um sinal de controle,

movendo sua alavanca para aquela posição estabelecida pelo controlador (PAMPLONA, 2018).

O servo possui três fios, como pode-se verificar na Figura 6, o vermelho que serve para entrada de energia, o marrom que tem a função de aterramento e o alaranjado, que deve ser ligado a uma porta do microcontrolador para receber instruções.

Figura 6 – TowerPro Microservo 9G SG90



Fonte: Autoria própria

2.4 APLICAÇÕES PARA DISPOSITIVOS MÓVEIS

Uma das principais características de um *smartphone* consiste em seu sistema operacional (SO). Recentemente (2018) o site IDC apresentou uma pesquisa demonstrando os percentuais de *Market Share* das principais marcas de sistemas operacionais e verifica-se os resultados na Tabela 1.

Tabela 1 - *Market Share* global de vendas entre os sistemas operacionais em 2016 a 2018

Ano	2016	2017	2018
Android	84,6%	85,1%	84,8%
iOS	14,7%	14,7%	15,1%
Outras	0,7%	0,2%	0,1%

Fonte: Adaptado de *Smartphone OS Market Share* – IDC (2018)

Após analisar-se a tabela acima, pode-se concluir que o Android lidera o mercado de *smartphones*, na sequência, o iOS com uma vasta diferença e por

último, em uma crescente queda, o Windows Phone praticamente sem capacidade de concorrer.

Dessa forma, desenvolver aplicativos para quaisquer outras plataformas que não sejam as duas primeiras, não é uma boa ideia. Para as empresas, deixa de ser uma opção lucrativa, porque gasta-se muito para manter um profissional com conhecimentos em diferentes tipos de linguagens para um baixo público alvo e, consequentemente, baixo lucro. Para os desenvolvedores, programar em plataformas nativas exige amplo conhecimento da IDE e API específica de cada SO, gasta-se então mais tempo dedicado aos estudos e cursos avançados (BONIATI; WAHLBRINCK, 2017).

Dentre as principais tecnologias existentes atualmente na área de programação móvel estão o desenvolvimento em ambiente nativo, o *WebApp* e o Multiplataforma.

2.4.1 Nativo

O primeiro passo para se criar um app nativo é decidir para qual sistema operacional (Android, iOS, Windows Phone) ele será desenvolvido. Ele é construído especificamente para uma única plataforma.

Um aplicativo nativo fica disponível na loja de aplicativos e permite que o usuário baixe e instale-o em seu próprio *smartphone*. Ele possui acesso a todos os recursos nativos do SO e sua principal vantagem é a utilização no modo *off-line*. Além disso, a experiência que o usuário terá em termos de *design* e segurança será melhor que todas as outras opções (FLING, 2009).

A sua principal desvantagem consiste na exclusividade de plataforma, o trabalho de conversão para outras plataformas é muito complexo e acaba se tornando muito caro para desenvolver, testar e prestar suporte a todas as versões (FLING, 2009).

2.4.2 *WebApp*

Os *WebApp*'s recebem esse nome porque eles consistem em *sites web* desenvolvidos para *desktop* que foram adaptados para uma versão móvel e a

segunda parte de seu nome faz referência a um aplicativo, porém ele não é instalado e nem compilado no dispositivo como os aplicativos nativos. O seu funcionamento ocorre através de navegadores, utilizando-se de seus recursos para conseguir executar seu conteúdo (FLING, 2009).

O seu ponto forte é que ele utiliza as linguagens de programação HTML, CSS e JavaScript, que são as mesmas de desenvolvimento *web*. E o seu principal ponto fraco é o desempenho, que é inferior ao nativo e demais formas (FLING, 2009).

2.4.3 Multiplataforma

Após verificar as duas formas acima, identificou-se que ambas possuem desvantagens, a primeira exige muito tempo de programação, e a segunda não possui acesso a recursos básicos necessários na maioria das aplicações.

Na tentativa de solucionar estes contratempos, o desenvolvimento multiplataforma permite que o desenvolvedor escreva somente um código fonte que poderá ser executado em múltiplos sistemas operacionais. Essa abordagem traz uma série de vantagens, como por exemplo o programador não precisa dominar mais de uma linguagem bem como suas API's e IDE's, ele elabora apenas um código, facilitando também a correção de erros (BERNARDES; MIYAKE, 2016).

O aplicativo multiplataforma, também chamado de Híbrido, pode ser baixado na loja de aplicativos e é instalado no dispositivo. Permite o acesso *off-line* e também o acesso as API's nativas, ou seja, possui acesso aos recursos nativos do celular como câmera, geolocalização, barômetro e outros (BERNARDES; MIYAKE, 2016).

2.4.3.1 Ionic

O Ionic Framework surgiu em 2012 com o intuito de facilitar a programação para dispositivos móveis. Atualmente em sua versão estável 3 e já com a versão 4 em fase de testes, o Ionic é um dos *frameworks* multiplataforma gratuitos mais populares do mundo. Seu código é aberto e possui uma enorme comunidade mundial (IONIC, 2018).

Sua linguagem é baseada em HTML e utiliza o CSS e JavaScript para personalizar e estilizar os componentes, proporcionando uma interface agradável ao usuário. A partir de outros *frameworks*, como Angular e Cordova, o Ionic consegue apresentar as informações de forma dinâmica e acessar os recursos nativos do aparelho (GRIFFITH, 2017).

O Angular é um projeto de código aberto mantido pela empresa Google, e é uma das ferramentas mais utilizadas em aplicações *web* devido seguir padrões de desenvolvimento para páginas *web* dinâmicas ocultando a complexidade do processo (GRIFFITH, 2017).

O Angular possui módulos específicos para facilitar a comunicação entre sistemas, como o HTTP, que é “um serviço básico do AngularJS que nos permite efetuar a comunicação com endpoints do servidor”, ou seja, permite ao programador enviar requisições através da URL para um servidor e assim extrair informações ou enviar comandos a serem executados por ele (GREEN; SESHADRI, 2014, p. 124).

Outro conceito importante do Angular é a injeção de dependência. Utiliza-se a injeção de dependência para diminuir os custos de manutenção de um sistema e focar no desenvolvimento das funcionalidades principais, pois ela funciona como uma classe que quando utilizada cria um objeto e o insere em partes do código através do construtor (YENER; THEEDOM, 2015).

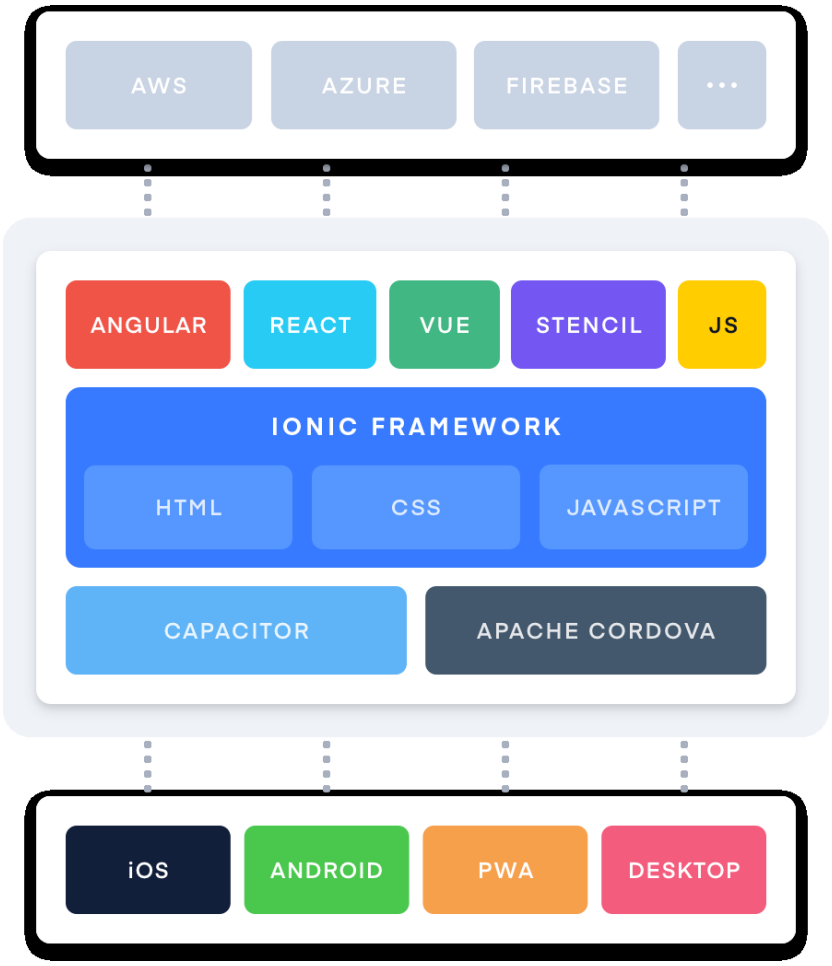
Por meio do Cordova, o Ionic consegue acessar as bibliotecas de recursos nativos do dispositivo e assim executar as funcionalidades O Cordova é mantido pela Apache, e faz o trabalho de tradução de linguagem *web* para a linguagem do SO, ou seja, realiza a integração entre os códigos HTML, CSS e JavaScript para acessar os recursos nativos do *device*, como câmera, acelerômetro, bússola, geolocalização, visualizar arquivos (RAVULAVARU, 2015).

Outra plataforma que também é utilizada com a mesma intenção é o Capacitor. Como descrito em seu próprio site, ele é um sucessor do Cordova e é um “conjunto consistente de APIs com foco na Web que permite que um aplicativo permaneça o mais próximo possível dos padrões Web, enquanto acessa recursos avançados de dispositivos nativos em plataformas que os suportam” (CAPACITOR, 2018, p. 1).

No Ionic, com apenas um código-fonte o aplicativo pode ser executado em múltiplas plataformas, tais como: Android, iOS, *WebApps* e *Desktop*, além disso pode-se integrar com diversos bancos de dados, como por exemplo: o AWS, Azure,

Firebase, dentre outros. Na Figura 7, exemplifica-se a estrutura do *framework* e apresenta ainda outras ferramentas às quais a plataforma se integra (React, VUE, Stencil) que possuem funções similares ao Angular (IONIC, 2018).

Figura 7 – Arquitetura Ionic



Fonte: Ionic (2018).

Para realizar a instalação do Ionic, se faz necessária a instalação do NodeJS, que "é uma plataforma construída sobre o motor JavaScript do Google Chrome para facilmente construir aplicações de rede rápidas e escaláveis", ou seja, ele gerencia a troca de informações entre cliente e servidor de forma eficiente e altamente escalável (NODEBR, 2016, p. 1).

Devido o *framework* Ionic ser gratuito e bastante utilizado na realidade atual de programação móvel, optou-se pela utilização do mesmo na elaboração do trabalho prático proposto. Um aplicativo simples, que pode ser desenvolvido em pouco tempo e com um resultado eficiente para multiplataformas, podendo ainda ser expandido facilmente caso haja necessidade futura (IONIC, 2018).

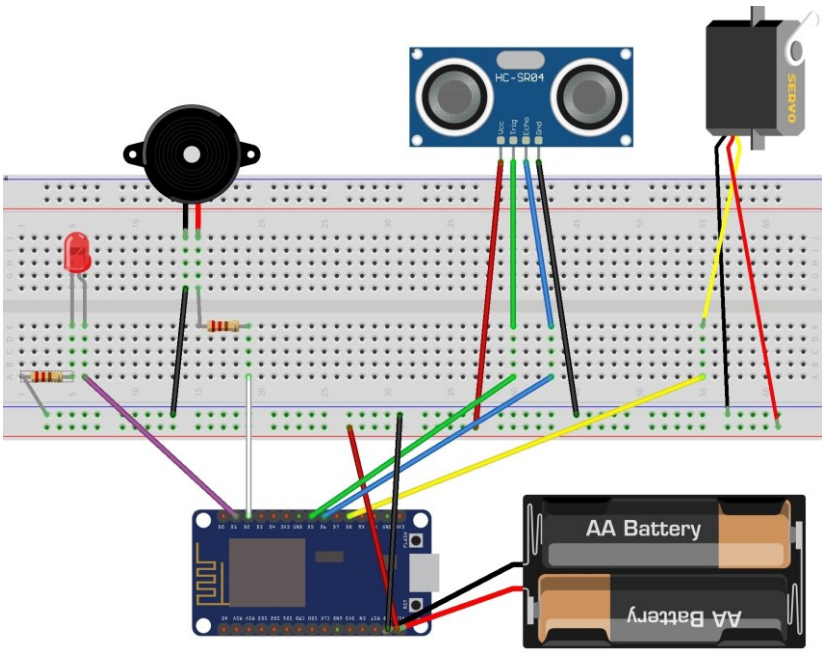
3 DESENVOLVIMENTO

A partir deste capítulo, retrata-se o trabalho em sua aplicação prática, apresentando as etapas desde a ligação física até a lógica implementada para comunicação entre os componentes e o código utilizado na aplicação móvel.

3.1 CONEXÕES FÍSICAS

Na Figura 7, demonstra-se como os componentes foram ligados fisicamente por meio da *protoboard*[1]. O componente principal do esquema é a placa ESP8266, ela recebe energia da fonte externa e a partir do código armazenado em sua memória interna, controla os dispositivos conectados. O Led irá acender-se ou apagar-se sempre que o botão for pressionado no aplicativo, o *buzzer* irá disparar-se sempre que o sensor ultrassônico detectar alguma presença ou quando o usuário apertar o botão de “disparar” no aplicativo. O servo irá girar de 0 a 180º, simulando a abertura e fechamento de uma porta, também mediante ao toque do usuário no botão.

Figura 8 – Esquema físico



Fonte: Autoria própria

[1] A *protoboard* ou matriz de contato é responsável pela interconexão dos componentes.

Baseando-se na sequência numérica da protoboard, a primeira das conexões realizadas foi a do LED. O lado negativo foi conectado ao GND, intermediado por um resistor de 330 ohm para limitar a corrente elétrica e evitar assim a queima do mesmo. Já o lado positivo foi ligado a porta digital D1 da placa.

Seguindo para a próxima conexão, tem-se o *Buzzer* também com duas entradas, uma positiva e outra negativa. A negativa foi conectada ao GND e o lado positivo, também intermediado por um resistor de 330 ohm, à porta digital D2 da placa.

Avança-se para a terceira conexão, que faz referência ao sensor ultrassônico. Para este dispositivo tem-se quatro conexões: VCC, TRIG, ECHO e GND. A primeira delas está conectada a porta VIN, a segunda e a terceira, às portas digitais D5 e D6, respectivamente e por último, o GND foi ligado ao GND da placa.

Concluindo os dispositivos utilizados no projeto, encontra-se o Servo Motor, com três conexões: alimentação, GND, entrada/saída. A alimentação foi ligada ao VIN, GND ao GND e entrada/saída à porta digital D8 da placa.

A placa ESP12E pode ser conectada tanto a uma fonte externa de energia, quanto a um dispositivo com entrada USB para garantir o seu funcionamento.

3.2 CÓDIGO ESP12E

Nesta seção será exposto o código carregado no microcontrolador para manipulação dos sensores e atuadores, e também para conexão com a rede.

Primeiramente na Figura 9, realizam-se as importações das bibliotecas para funcionamento do recurso de WiFi e também para manipulação do servo motor. Nas linhas 4-7, definem-se as portas digitais para cada um dos componentes, com exceção do Servo, que será definido adiante. Na linha 9 cria-se o objeto do tipo Servo, utilizando a biblioteca previamente importada. Nas linhas 11-15, criam-se as variáveis auxiliares para lógica do código, mais especificamente no controle do alarme. Nas linhas 17 e 18, inserem-se o nome e senha da rede. Mais adiante, nas linhas 20-23, definem-se através da biblioteca ESP8266 WiFi: IP fixo, gateway, máscara e porta da rede, respectivamente.

Na sequência, na Figura 10, abre-se uma estrutura de repetição para configuração inicial dos dispositivos conectados a placa. Nas linhas 26-29, escolhe-se qual será o fluxo da informação nas portas digitais, podem assumir o valor booleano de "*OUTPUT"* para saída ou "*INPUT"* para entrada. Na linha 31, finalmente se define a porta digital escolhida para o servo. Nas linhas 33 e 35, escolhe-se o valor inicial para o LED como *LOW* (desligado) e o servo assume o ângulo de 180º.

Ainda na mesma estrutura de repetição, na linha 38 da Figura 11, escolhe-se a porta Serial utilizada para comunicação. Nela serão apresentadas informações de quando o dispositivo estiver se conectando e também o nome da rede (linhas 41-44). Então envia-se o "ssid" e senha juntamente com as configurações de "ip", "gateway" e "subnet" para se conectar à rede. Nas linhas 50-53, o código apresenta um laço *while* que verifica o resultado da função "WiFi.status()" e permanece iterando enquanto for diferente de "WL_CONNECTED". Assim que conectado, é comunicado pela porta Serial o sucesso através das linhas 54 e 55. Na sequência utiliza-se a função para iniciar o server, logo em seguida também é impresso o sucesso e qual o endereço http para acesso ao mesmo.

Na Figura 12, linha 65, inicia-se um novo laço de iteração que contém a lógica de controle dos sensores e atuadores. Nas linhas 66-71, é iniciado o sensor ultrassônico. A porta "Trig" é definida como *HIGH* (ligada), ocorre um pequeno *delay* e então a porta é definida como *LOW* (desligada). Na sequência, a variável "duracao" recebe um pulso da porta "Echo" e logo em seguida, a variável é inserida na formula de conversão para centímetros. É importante destacar-se que estas instruções pertencem a este *loop* devido a verificação do sensor ultrassônico ser em tempo real, não podendo ser inicializadas no início do código como valor estático.

Ainda na Figura 12, nas linhas 73-75 trata-se do funcionamento do alarme em tempo real, que também precisa estar no começo do laço pelo mesmo motivo das instruções anteriores. O *if* verifica se duas condições são verdadeiras. A primeira é se a variável "ativar" está atualmente como *true*, esta variável só é definida com esse valor booleano em outro momento do código a partir de uma escolha do usuário na interface do aplicativo. A segunda condição, verifica se o valor da variável "distancia", que monitora em tempo real a existência de algum objeto no perímetro do sensor, está entre 0 e 40 centímetros. Caso ambas condições sejam verdadeiras, então chama-se a função "dispara()" que será explicada adiante.

Após a verificação do alarme, na Figura 13, nas linhas 77-80 cria-se objeto "*client*" do tipo "WiFiClient", que recebe a informação e verifica se o servidor está disponível, caso não esteja, retorna para o laço anterior para que seja reconectado à rede e reiniciado o servidor. Nas linhas 81-84, existe um laço que aguarda até que o cliente envie alguma requisição, assim que ele enviar, nas linhas 86-88 o controlador irá armazenar o valor em uma variável do tipo *String*, imprimir na porta serial e enviar um *flush* para limpar a requisição do cliente.

Com a requisição armazenada na variável, iniciam-se as verificações para execução das ações solicitadas por meio de condições "*if*". A primeira delas é o servo nas linhas 91-96 da Figura 12, caso o valor da requisição seja abrir ou fechar, o servo motor irá abrir ou fechar a porta, com valor 0 ou 180 na função "servo.write()", respectivamente. Nas linhas 99-104 da mesma figura, analisa-se se a requisição recebida é acender ou apagar, então envia-se à porta digital do LED os comandos *HIGH* ou *LOW*.

Ainda analisando a requisição do cliente, nas linhas 106-112 da Figura 14, analisa-se se a requisição recebida foi "ativar" ou "desativar". O sistema irá ligar ou desligar o funcionamento do alarme, é também onde define-se o valor booleano da variável "ativar" mencionado anteriormente, como *true* ou *false*. Nas linhas 113-115 faz-se uma verificação da distância, para que o sistema não dispare assim que ativado.

Nas linhas 118-120 da Figura 14, o sistema apresenta um recurso que será disponibilizado na interface, que seria semelhante a uma emergência onde o usuário deseja disparar o alarme por algum motivo. O sistema então irá aguardar a requisição "disparar" e então irá chamar a função "dispara()".

Por fim, na Figura 15 a função "dispara()" inicia-se declarando a variável "cont" com valor 0 (limita os disparos a três repetições), liga-se o led, define-se o ângulo do servo motor para 180 (representando a porta fechada) e dispara o *buzzer.* O som é disparado através do laço *for*, que inicia com a frequência 150, iterando até 1800, a cada volta, a função *tone* envia sinal para porta digital do dispositivo, com a respectiva frequência e tempo. Logo em seguida existe o mesmo laço iterando de forma inversa, desta forma o ruído emitido se assemelha ao de uma sirene.

3.3 APLICAÇÃO MÓVEL IONIC

A elaboração do aplicativo móvel baseou-se o framework Ionic, o qual permite desenvolver apenas um código para mais de uma plataforma. Para começar um projeto Ionic, deve-se navegar até a pasta que irá utilizar através do NodeJS e digitar o comando “ionic start –type=angular” para que o mesmo utilize a versão 4 (versão beta). Em seguida o terminal irá questionar se deseja utilizar a versão 4, o nome do projeto, o *template* (blank, sidemenu ou tabs), se deseja integra-lo ao Cordova e se deseja instalar a versão Pro do SDK.

Para este protótipo, utilizou-se o *template* “sidemenu”, que possui uma estrutura com um menu lateral que possibilita acesso a outras páginas do app e o nome escolhido foi “ArdoHouse”.

A primeira tela do aplicativo é a tela de *login*, para o cadastro e validação de contas, realizou-se a integração com a plataforma Firebase, que controla os dados de cadastro e permite o controle de acesso. Para utilizar-se destes serviços, é necessário instalar a SDK do Firebase, através do terminal do NodeJS com o comando “npm install firebase --save".

Após isso, criam-se os serviços, como pode-se verificar na Figura 16, que são utilizados em todas as páginas para controle do login, também através do terminal com os comandos “ionic g service services/firebase” e “ionic g service services/usuario”.

Após criados os serviços, ainda na Figura 16, realizam-se as importações dos recursos utilizados e no construtor, definem-se as configurações que são geradas automaticamente pelo Firebase e devem ser coletadas no site para então inicializar o app com as devidas configurações. Então, criam-se os métodos “db” e “auth” que chamam os métodos do servidor. O banco de dados não foi implementado neste trabalho, mas foi criado para facilitar futuras expansões no aplicativo.

O outro serviço criado controla o usuário, verificando se o mesmo permanece logado através de um observador, como mostra a Figura 17. Com o método “onAuthStateChanged” verifica-se de o usuário contém informações. Caso seja verdade, o “usuario” recebe o “userId” e “email” do usuário logado, caso não, recebe valor nulo.

Na Figura 18, nas linhas 1-5, definiu-se o *header* (cabeçalho) da página através do componente “ion-toolbar” com o atributo de cor “primary” (nome padrão da linguagem que faz referência a cor azul) e título da página como “Login”.

O conteúdo visual da página pode-se verificar na Figura 19, e se inicia com o componente “ion-item” que recebe o *input* de e-mail, e em seguida outro componente de mesmo formato para receber a senha. Após isso utilizam-se dois “ion-button” que possuem uma ação de acordo com o clique, chamando as funções de “login” e de “criarConta”, para ambos se utilizou o “ion-icon” que permite adicionar ícones nos componentes.

Por fim, foi criado outro item, para armazenar uma informação de que a senha deve conter 6 dígitos. Este componente só será visualizado na tela se o *input* de senha estiver com menos de 6 caracteres, interação que foi define-se pela condição estabelecida no “*ngIf”.

Na Figura 20, nas duas primeiras linhas realiza-se a importação dos componentes externos que são utilizados na página. Nas linhas 10 e 11, criam-se as variáveis de “email” e “senha” que são utilizadas nos *inputs* do HTML. Após isso, criam-se duas funções assíncronas que são chamadas pelo clique dos botões da interface. A função “login” coleta os dados escritos nos *inputs* e envia como parâmetro para função “singInWithEmailAndPassword” da plataforma Firebase, onde os dados são validados e assim torna-se possível o acesso. Já a função de “criarConta”, faz o mesmo procedimento, porém envia parâmetros para a função “createUserWithEmailAndPassword” da plataforma Firebase.

Após realizar o acesso, o aplicativo redireciona para a página “home”, que é a página principal, como verifica-se na Figura 21.

Nas linhas 1-15 da Figura 22 foi definido o *header*, primeiramente foram definidos “ngIf’s” para que a página não seja exibida caso a variável “email” fosse nula com um componente “ion-toolbar” com a cor azul e com o ícone definido na posição inicial do componente pela propriedade “slot = start”. O título da página é o nome do aplicativo “ArdoHouse” e adicionou-se um botão para realizar o “logout” na posição final do componente. O conteúdo da página apresenta o componente “ion-slide” que contém quatro “ion-card’s” com imagem e texto para o usuário deslizar, e obter informações sobre os recursos do aplicativo.

Na Figura 23, inicialmente são realizadas as importações dos recursos, então definidos os componentes, e na linha 14 criada variável “email”. No construtor

são injetados os componentes e nas linhas 20 e 21 coleta-se o e-mail do usuário logado. Tem-se como resultado a tela da Figura 24.

O menu lateral constitui-se do componente “ion-menu”, como demonstrado na Figura 25, que somente é exibido se existir usuário conectado. No cabeçalho, apresenta-se a foto do usuário e seu e-mail e no corpo da página, realiza-se um “*ngFor” que verifica todas as páginas cadastradas no *array* “appPages” apresentando-as com seus respectivos nomes e caso usuário clique, será redirecionado para aquele endereço.

Nas configurações de rotas do aplicativo, nas linhas 4-17 da Figura 26, definem-se os endereços de todas as páginas criadas no protótipo. A navegação entre páginas é realizada através de especificação de rotas, ou seja, caminhos que levam outros endereços. Cada uma das opções apresentadas dentro do “*ngFor” possui uma URL, que ao clicar redirecionam a visualização para outra página, fechando o menu lateral e sobrepondo a página principal da aplicação como pode-se verificar na Figura 27.

Na Figura 28, após se realizar a importação dos serviços e criação das variáveis principais, no construtor, injetam-se os serviços e coletam-se os dados do usuário logado. Ao iniciar o app, realiza-se uma verificação através do método “onAuthStateChanged” do Firebase se o usuário está ativo, se estiver, é direcionado para página “home”, caso não esteja, será redirecionado para página de “login”.

No menu lateral, apresentam-se as páginas criadas neste projeto para controle de cada um dos recursos conectados ao microcontrolador. A página de “Lista de compras” foi criada para implementação futura e, portanto, não será exibida nas seguintes explicações.

A primeira página de controle criada foi a do “Alarme”, na Figura 29 verifica-se a parte visual desta página e na Figura 30 o código-fonte. Nas linhas 1-6 apresenta-se o seu cabeçalho, onde segue-se o padrão da tela “Home”, porém sem o botão de “logout”. Em seu conteúdo, apresenta-se um componente “ion-card” que possui uma imagem e três botões (Ativar, Desativar e Disparar), os três possuem um evento do clique que chama a função “acao” enviando como parâmetro sua respectiva ação. Embaixo do botão disparar, apresenta-se um item com um texto alertando usuário que este botão só deve ser disparado em casos de emergência. Quando o usuário pressiona qualquer uma das teclas de ação, o sistema emite uma tela pedindo a confirmação da ação, como verifica-se na Figura 31.

Na Figura 32, inicialmente, realizam-se as importações dos serviços, e captura-se o e-mail do usuário para controle da página. Então na Figura 33, a função de “acao” recebe o parâmetro enviado e cria uma constante com nome “actionSheet” que recebe as configurações do “actionSheetController”. No *controller*, são criados o título da modal, os botões e as ações que se executam conforme a escolha do usuário e por fim exibe a modal. Caso o usuário confirme esta ação, será chamada outra função “solicitar” juntamente com o parâmetro recebido pelo clique do botão na interface. A função “solicitar” recebe o parâmetro e realiza uma requisição ao servidor pelo método “http.get”, enviando a url mais o parâmetro.

A tela de controle de Luzes, pode-se verificar na Figura 34, e segue o mesmo padrão que a tela de Alarme, porém só possui dois botões, o de Acender e o de Apagar e seu código fonte pode-se verificar na Figura 35. Também possui a confirmação de ação e quando se confirma, envia-se o parâmetro da ação para a função “acao” que irá enviar requisição ao servidor, assim como na tela de Alarme.

A tela de controle de Portas segue o mesmo padrão que a tela de Alarme e Luzes, somente dois botões, o de Abrir e o de Fechar e pode-se verificar na Figura 36. Também possui a confirmação de ação e quando se confirma, envia-se o parâmetro da ação para a função “acao” que irá enviar requisição ao servidor, assim como na tela de Alarme e Luzes e seu código-fonte verifica-se na Figura 37.

Por fim, a tela “Sobre”, que se pode verificar na Figura 38, onde apresentam-se informações de contato sobre o desenvolvedor do aplicativo através dos componentes “ion-card” contendo um “ion-list” que por sua vez engloba um “ion-item” com um “ion-avatar” e “ion-button’s” para redirecionamento para outros sites. O código-fonte apresenta-se na Figura 39.

4 CONCLUSÃO

Com a elaboração deste trabalho pode-se concluir que a Internet das Coisas já está presente em nossas vidas, o mercado possui inúmeros microcontroladores de várias marcas e que possibilitam a configuração de servidores e, consequentemente, a comunicação via rede *wireless*.

Agregando a estes dispositivos os sensores ou atuadores, pode-se utilizar a criatividade para inventar novos objetos inteligentes, permitindo a inovação tanto nas residências, como nas empresas e indústrias.

O homem da atualidade possui inúmeras atividades no seu cotidiano e está constantemente recebendo informações e tomando decisões no seu *smartphone*. A cada dia se torna mais importante desenvolver novos aplicativos para atender as mais diversas necessidades desta nova forma de vida.

Após finalizar esta pesquisa, verificou-se que existem muitas ferramentas disponíveis para a criação e estilização de aplicações móveis. Os *frameworks* facilitam o desenvolvimento minimizando a complexidade que há por trás da criação de um sistema. Pode-se criar um aplicativo gastando pouco tempo e com pouco investimento, atraindo os olhares de desenvolvedores e empresas.

Desenvolveu-se neste trabalho um protótipo de automação residencial com base nos objetivos específicos propostos. Primeiramente, realizou-se a pesquisa sobre o ESP12E, bem como sua integração com sensores, atuadores e Internet. Após isso, criou-se o código para funcionamento de cada dispositivo conectado ao microcontrolador, que seriam ativados ou desativados conforme requisições a serem enviadas ao servidor.

Por fim, elaborou-se o aplicativo multiplataforma, que integra o seu funcionamento com o microcontrolador e é quem envia as requisições através de seus botões.

Utilizou-se ainda, uma maquete para representar a casa com todos os componentes posicionados e assim demonstrar o sistema em funcionamento.

As principais dificuldades encontradas na elaboração deste trabalho ocorreram em dois pontos, inicialmente na criação do código utilizado no microcontrolador, onde vários testes foram necessários até que se chegasse na harmonia entre os componentes. Em segundo lugar, na elaboração do aplicativo,

onde a complexidade foi maior devido a utilização de conceitos de diversos *frameworks* que são integrados ao Ionic.

Uma das limitações deste trabalho se deve ao fato de o aplicativo não ser parametrizável, para isso deve-se alterar o código-fonte. Pode-se citar também outra limitação que é a falta de implementação o banco de dados, onde poderiam se armazenar e fornecer informações à aplicação. Ainda sobre as limitações, não é possível saber o atual *status* dos componentes, por exemplo, se a porta está aberta ou fechada, se a luz está acesa ou apagada, ou se o alarme está ativo.

4.1 TRABALHOS FUTUROS

A inteligência artificial está bastante ligada a *IoT*, neste trabalho controlou-se uma luz, uma porta e um alarme, e pode-se tranquilamente aprimorar a forma de controle, tornando as tarefas automáticas. Por exemplo, definir-se que todas as luzes da casa irão se apagar e o alarme será ativado, após o usuário deitar na cama. Com a implementação de um banco de dados que armazene informações sobre as ações do usuário no app, podem ser realizadas análises estatísticas e, assim, definir horário em que o usuário dorme e acorda automaticamente.

O banco de dados poderia apresentar outras vantagens que podem ser abordadas em trabalhos futuros, como por exemplo a emissão de relatórios, tornando possível verificar os pontos em que mais se gasta energia elétrica, quais hábitos estão relacionados ao desperdício e também estabelecer metas para consumo gerando economia e sustentabilidade para a residência.

Além disso, o aplicativo pode ser expandido para comportar outros recursos de administração residencial, como por exemplo finanças pessoais. Um módulo que permitiria um controle de fluxo de caixa simples, com integração à conta bancária, extraindo facilmente as entradas e saídas de dinheiro e com isso aproximando os gastos e possibilitando ao responsável pela família tomar decisões com base em informações reais de suas necessidades básicas.

REFERÊNCIAS

ALVES, R. M.; et al. Uso do Hardware Livre Arduino em Ambientes de Ensino-aprendizagem. In: CONGRESSO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO. 1., 2012, Rio de Janeiro. **Anais da Jornada de Atualização em Informática na Educação.** Rio de Janeiro: UF-RJ; UNIRIO, 2012. p. 162-187.

BERNARDES, T. F.; MIYAKE, M. Y. Cross-platform Mobile Development Approaches: A Systematic Review. **Revista IEEE América Latina,** São Paulo (SP), v. 14, n. 4, p. 1892-1898, jun. 2016.

BONIATI, B. B.; WAHLBRINCK, K. A. Aplicações mobile híbridas: um estudo de caso do framework Ionic para construção de um diário de classe. In: ENCONTRO ANUAL DE TECNOLOGIA DA INFORMAÇÃO. 8., 2017, Frederico Westphalen. **Anais do EATI.** Frederico Westphalen: IFFCFW-RS, 2017. p. 69-76.

CAPACITOR. Capacitor: Native progressive web Apps. 2018. Disponível em: <https://capacitor.ionicframework.com/docs/>. Acesso em: 02 dez. 2018.

DELOITTE, T. T. **Global Mobile Consumer Survey 2017**. O celular no dia a dia do brasileiro. 2007. Disponível em: <https://www2.deloitte.com/content/dam/Deloitte/br/Documents/technology-media-telecommunications/Global-Mobile-Consumer-survey-2017.pdf>. Acesso em: 23 jun. 2018.

DUTRA, M. S.; ROMANO, V. F. **Introdução à robótica industrial**. 1. ed. São Paulo: Blucher, 2016.

ERTHAL, A. A. O telefone celular como produtor de novas sensorialidades e técnicas corporais. **Revista Contemporânea,** Rio de Janeiro (RJ), v. 5, n. 8, p. 58-65, 2007.

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

FLING, B. **Mobile Design and Development**. 1. ed. Estados Unidos da América: O'Reilly Media, 2009.

FONSECA FILHO, C. **História da Computação:** O caminho do pensamento e da tecnologia. 1. ed. Porto Alegre: EDIPUCRS, 2007.

GRIFFITH, C. **Mobile app development with Ionic:** Cross-platform apps with Ionic, Angular & Cordova. 1. ed. Estados Unidos da América: O'Reilly Media, 2017.

IDC. **Smartphone OS Market Share**. 2018. Disponível em: <https://www.idc.com/promo/smartphone-market-share/os>. Acesso em: 13 out. 2018.

IONIC. **All about Ionic**. 2018. Disponível em: <https://ionicframework.com/about>. Acesso em: 23 jun. 2018.

MONK, S. **Programação com Arduino:** Começando com Sketches. 1. ed. Bookman, 2013.

NODEBR. O que é Node.js?. 14 nov. 2016. Disponível em: <http://nodebr.com/o-que-e-node-js/>. Acesso em: 02 dez. 2018.

OLIVEIRA, S. **Internet das coisas com ESP8266, Arduino e Raspberry PI**. 1. ed. São Paulo: Novatec, 2017.

PAMPLONA, Y. L. **Indústria 4.0:** Análise de uma nova era industrial. 2018. 49 f. Dissertação – Graduação em Sistemas de Informação, Universidade Federal do Estado do Rio de Janeiro. Rio de Janeiro, 2018.

RAVULAVARU, A. **Learning Ionic:** Buid real-time and hybrid mobile applications with Ionic. 1. ed. Birmingham: Packt Publishing, 2015.

REVISTA EXAME. **Omo cria prendedor de roupas que avisa quando vai chover**. 20 abr. 2016. Disponível em <http://exame.abril.com.br/marketing/noticias/omo-cria-prendedor-de-roupas-que-avisa-quando-vai-chover>. Acesso em: 23 jun. 2018.

SANTAELLA, L. Mídias locativas: A internet móvel de lugares e coisas. **Revista Famecos**, Porto Alegre (RS), v. 1, n. 35, p. 95-101, fev. 2015. Disponível em <http://docslide.com.br/documents/inguagens-liquidas-na-era-da-mobilidade.html>. Acesso em: 23 jun. 2018.

SANTOS, B. P.; et al. Livro texto: Minicursos. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS. 34., 2016, Salvador. **Anais do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos.** Salvador: UF-BA, 2016. p. 1-50.

SESHADRI, S.; GREEN, B. **Desenvolvendo com AngularJS:** aumento de produtividade com aplicações web estruturadas. 1. ed. São Paulo: Novatec, 2014.

SOUZA, M. V. de. **Domótica de baixo custo usando princípios de IoT**. 2016. 50 f. Dissertação – Programa de Pós-Graduação em Engenharia de Software, Universidade Federal do Rio Grande do Norte. Natal, 2016.

WENDLING, M. **Sensores.** 8 jan. 2016. Disponível em: <http://www2.feg.unesp.br/Home/PaginasPessoais/ProfMarceloWendling/4---sensores-v2.0.pdf>. Acesso em: 23 jun. 2018.

YENER, M.; THEEDOM, A. **Professional:** Java EE Design Patterns. 1. ed. Estados Unidos da América: Wrox, 2015.

APÊNDICE A - Código-fonte do microcontrolador

Figura 9 – Código ESP8266 – Parte 1

```
#include <ESP8266WiFi.h> //Biblioteca WiFi
#include <Servo.h> //Biblioteca Servo

#define ledPorta 5 //Porta digital D1 no NodeMCU
#define buzzerPorta 4 //Porta digital D2 no NodeMCU
#define trigPorta 14 //Porta digital D5 no NodeMCU
#define echoPorta 12 //Porta digital D6 no NodeMCU

Servo servo;//Objeto do tipo Servo

int tempo = 10; //Variável para utilização na lógica do sensor ultrassônico
int frequencia = 0; //Variável para utilização na lógica do sensor ultrassônico
int duracao, distancia; //Variável para utilização na lógica do sensor ultrassônico
int cont = 0; //Variável para utilização no controle do alarme
bool ativar = false; //Variável para utilização na lógica do alarme

const char* ssid = "";//Nome da rede
const char* password = "";//Senha da rede

IPAddress ip(192,168,15,110); //Define o IP fixo
IPAddress gateway(192,168,15,1); //Gateway da rede
IPAddress subnet(255,255,255,0); //Máscara da rede
WiFiServer server(80);//Porta da rede
```

Fonte: Autoria própria

Figura 10 – Código ESP8266 – Parte 2

```
void setup() {
  pinMode(ledPorta, OUTPUT); //Porta digital como saída/entrada de informações
  pinMode(buzzerPorta, OUTPUT);
  pinMode(trigPorta, OUTPUT);
  pinMode(echoPorta, INPUT);

  servo.attach(15); //Define porta digital para o servo D8 na NodeMCU

  digitalWrite(ledPorta, LOW); //Seta status inicial do led como desligado

  servo.write(180);
```

Fonte: Autoria própria

Figura 11 – Código NODE MCU – Parte 3

```
  //Conexão
  Serial.begin(115200); //Define da porta Serial de comunicação
  delay(10);

  Serial.println();
  Serial.println();
  Serial.print("Conectando na rede ");
  Serial.println(ssid);

  WiFi.begin(ssid, password); //Envia id e senha da rede para função da biblioteca wifi
  WiFi.config(ip, gateway, subnet); //Envia parâmetros para função que irá configurar a rede

 //Cria um laço que compara resultado da função status
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi conectado");

  server.begin();//Inicia server
  Serial.println("Servidor iniciado");
  Serial.print("Use este URL para conectar: ");
  Serial.print("http://");
  Serial.print(WiFi.localIP());
  Serial.println("/");
}
```

Fonte: Autoria própria

Figura 12 – Código NODE MCU – Parte 4

```
void loop() {
  digitalWrite(trigPorta, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPorta, LOW);
  duracao = pulseIn(echoPorta, HIGH);
  distancia = (duracao / 2) / 29.1; //Define base do cálculo de conversão
  delay(10);

  if (ativar==true && (distancia < 40 && distancia > 0)) {
    dispara();
  }
```

Fonte: Autoria própria

Figura 13 – Código NODE MCU – Parte 5

```
77  WiFiClient client = server.available(); //Verifica se o cliente está conectado
78  if (!client) {
79    return;
80  }
81  Serial.println("Novo cliente se conectou!"); //Espera até o cliente enviar algum dado
82  while(!client.available()){
83    delay(1);
84  }
85
86  String request = client.readStringUntil('\r'); //Lê a primeira linha da requisição
87  Serial.println(request);
88  client.flush();
89
90  //Servo
91  if (request.indexOf("/abrir") != -1)  {
92    servo.write(0); //Move o servo para 0 graus
93  }
94  if (request.indexOf("/fechar") != -1)  {
95    servo.write(180); //Move o servo para 180 graus
96  }
97
98  //Led
99  if (request.indexOf("/acender") != -1)  {
100   digitalWrite(ledPorta, HIGH); //Escreve no led LIGADO
101 }
102 if (request.indexOf("/apagar") != -1)  {
103   digitalWrite(ledPorta, LOW); //Escreve no led DESLIGADO
104 }
```

Fonte: Autoria própria

Figura 14 – Código NODE MCU – Parte 6

```
106   //Ativar/Desativar alarme
107   if (request.indexOf("/ativar") != -1)  {
108     ativar = true;
109   }
110   if (request.indexOf("/desativar") != -1)  {
111     ativar = false;
112   }
113   if (ativar==true && (distancia >= 40 || distancia <= 0)) {
114     digitalWrite(buzzerPorta, LOW);
115   }
116
117   //Disparar
118   if (request.indexOf("/disparar") != -1)  {
119       dispara();
120   }
121 }
```

Fonte: Autoria própria

Figura 15 – Código NODE MCU – Parte 7

```
123 //Função que dispara o alarme
124 void dispara(){
125   cont = 0;
126   digitalWrite(ledPorta, HIGH);
127   servo.write(180);
128   while (cont<=2) {
129     for (frequencia = 150; frequencia < 1800; frequencia += 1) { //Tone que produz sirene de polícia
130     tone(buzzerPorta, frequencia, tempo);
131     delay(3);
132     }
133     for (frequencia = 1800; frequencia > 150; frequencia -= 1) { //Tone que produz sirene de polícia
134     tone(buzzerPorta, frequencia, tempo);
135     delay(3);
136     }
137     cont++;
138   }
139 }
```

Fonte: Autoria própria

APÊNDICE B - Código-fonte do aplicativo móvel

Figura 16 – Código Ionic – Firebase Service TypeScript

```typescript
import { Injectable } from '@angular/core';
import * as firebase from 'firebase/app';
import 'firebase/firestore';
import 'firebase/auth';

@Injectable({
  providedIn: 'root'
})
export class FirebaseService {

  constructor() {
    const config = {
      apiKey: "",
      authDomain: "",
      databaseURL: "",
      projectId: "",
      storageBucket: "",
      messagingSenderId: ""
    };
    firebase.initializeApp(config);
  }

  db() {
    return firebase.firestore();
  }

  auth(){
    return firebase.auth();
  }
}
```

Fonte: Autoria própria (2018)

Figura 17 – Código Ionic – Usuario Service TypeScript

```typescript
import { Injectable } from '@angular/core';
import { FirebaseService } from './firebase.service';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class UsuarioService {

  constructor(private firebase: FirebaseService) { }

  getUser() {
    let usuario;
    return Observable.create(observer => {
      this.firebase.auth().onAuthStateChanged(user => {
        if (user) {
          usuario = { userId: user.uid, email: user.email };
        } else {
          usuario = { userId: '', email: '' };
        }
        observer.next(usuario);
      });
    });
  }
}
```

Fonte: Autoria própria (2018)

Figura 18 – Código Ionic – Página de Login HTML

```
<ion-header>
  <ion-toolbar color="primary">
    <ion-title text-center>Login</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content padding>

  <ion-item>
    <ion-label position="floating">Email</ion-label>
    <ion-input [(ngModel)]="email"></ion-input>
  </ion-item>
  <ion-item>
    <ion-label position="floating">Senha</ion-label>
    <ion-input [(ngModel)]="senha" type="password" ></ion-input>
  </ion-item>

  <ion-button expand="block" (click)="login()"><ion-icon name="lock" slot="start"></ion-icon>Login</ion-button>
  <ion-button expand="block" fill="outline" (click)="criarConta()">
    <ion-icon name="person" slot="start"></ion-icon>
    Registrar
  </ion-button>

  <ion-item *ngIf="senha.length < 6">
    <ion-icon name="information"></ion-icon>
    <ion-item >
      <p>A senha deve conter 6 dígitos</p>
    </ion-item>
  </ion-item>
</ion-content>
```

Fonte: Autoria própria (2018)

Figura 20 – Código Ionic – Página de Login TypeScript

```
import { Component, OnInit } from '@angular/core';
import {FirebaseService} from './../services/firebase.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.page.html',
  styleUrls: ['./login.page.scss'],
})
export class LoginPage implements OnInit {
  public email = '';
  public senha = '';

  constructor(private firebase: FirebaseService) { }
  ngOnInit() {
  }

  async login() {
    try{
      await this.firebase.auth().signInWithEmailAndPassword(this.email, this.senha);
    } catch (e) {
      throw new Error(e);
    }
  }

  async criarConta() {
    try{
      await this.firebase.auth().createUserWithEmailAndPassword(this.email, this.senha);
    } catch (e) {
      throw new Error(e);
    }
  }
}
```

Fonte: Autoria própria (2018)

Figura 22 – Código Ionic – Página Home HTML

```
<ion-header *ngIf="!email==''">
    <ion-toolbar color="primary">
      <ion-buttons slot="start">
        <ion-menu-button></ion-menu-button>
      </ion-buttons>
      <ion-title>
        ArdoHouse
      </ion-title>
      <ion-buttons slot="end">
        <ion-button>
            <ion-icon name="log-out"></ion-icon>
        </ion-button>
      </ion-buttons>
    </ion-toolbar>
</ion-header>

<ion-content padding text-center *ngIf="!email==''">
    <ion-slides pager="true">
        <ion-slide>
            <ion-card  color="light">
                <ion-img src="../../assets/imagem1.jpg"></ion-img>
                <ion-card-header>
                    <ion-card-title>Explore o aplicativo!</ion-card-title>
                </ion-card-header>
                <ion-card-content>
                    <p>Para iniciar acesse o menu lateral.</p>
                </ion-card-content>
                <br><br>
            </ion-card>
        </ion-slide>
        <ion-slide>
```

Fonte: Autoria própria (2018)

Figura 23 – Código Ionic – Página Home TypeScript

```
import { Component } from '@angular/core';
import { App, MenuController, NavController} from "@ionic/angular";
import { FirebaseService } from '../services/firebase.service';
import { UsuarioService } from '../services/usuario.service';
import { Platform } from '@ionic/angular';

@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
export class HomePage {

  public email;

  constructor(public nav: NavController, public app: App,
              public menuCtrl: MenuController, private firebase: FirebaseService,
              private usuario: UsuarioService, private platform: Platform){
    this.platform = platform;
    this.usuario.getUser().subscribe(user => {
      this.email = user.email;
    });
  }

  ngOnInit() {
  }
}
```

Fonte: Autoria própria (2018)

Figura 25 – Código Ionic – Menu lateral HTML

```
<ion-app >
  <ion-split-pane>
    <ion-menu *ngIf="!email==''">
      <ion-header>
        <ion-toolbar color="primary">
          <ion-avatar slot="start">
            <ion-img [src]="foto"> </ion-img>
          </ion-avatar>
          <ion-title>{{email}}</ion-title>
        </ion-toolbar>
      </ion-header>
      <ion-content>
        <ion-list>
          <ion-menu-toggle auto-hide="false" *ngFor="let p of appPages">
            <ion-item [routerDirection]="'root'" [routerLink]="[p.url]">
              <ion-icon slot="start" [name]="p.icon"></ion-icon>
              <ion-label>
                {{p.title}}
              </ion-label>
            </ion-item>
          </ion-menu-toggle>
        </ion-list>
      </ion-content>
    </ion-menu>
    <ion-router-outlet main></ion-router-outlet>
  </ion-split-pane>
</ion-app>
```

Fonte: Autoria própria (2018)

Figura 26 – Código Ionic – Menu lateral Rotas

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [
  {
    path: '',
    redirectTo: 'login',
    pathMatch: 'full'
  },
  { path: 'login', loadChildren: './login/login.module#LoginPageModule' },
  { path: 'home', loadChildren: './home/home.module#HomePageModule' },
  { path: 'lista', loadChildren: './list/list.module#ListPageModule' },
  { path: 'alarme', loadChildren: './alarme/alarme.module#AlarmePageModule' },
  { path: 'luz', loadChildren: './luz/luz.module#LuzPageModule' },
  { path: 'porta', loadChildren: './porta/porta.module#PortaPageModule' },
  { path: 'sobre', loadChildren: './sobre/sobre.module#SobrePageModule' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

Fonte: Autoria própria (2018)

Figura 27 – Código Ionic – Menu lateral TypeScript – Parte 1

```typescript
import { Component } from '@angular/core';
import { Platform } from '@ionic/angular';
import { SplashScreen } from '@ionic-native/splash-screen/ngx';
import { StatusBar } from '@ionic-native/status-bar/ngx';
import { FirebaseService } from './services/firebase.service';
import { Router } from '@angular/router';
import { UsuarioService } from './services/usuario.service';

@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html'
})

export class AppComponent {
  public appPages = [
    { title: 'Início', url: '/home', icon: 'home' },
    { title: 'Lista de compras', url: '/lista', icon: 'basket' },
    { title: 'Alarme', url: '/alarme', icon: 'eye' },
    { title: 'Luzes', url: '/luz', icon: 'bulb' },
    { title: 'Portas', url: '/porta', icon: 'key' },
    { title: 'Contato', url: '/sobre', icon: 'code-working' }
  ];

  public email;
  public foto;
```

Fonte: Autoria própria (2018)

Figura 28 – Código Ionic – Menu lateral TypeScript – Parte 2

```typescript
  constructor(
    private platform: Platform, private splashScreen: SplashScreen,
    private statusBar: StatusBar, private firebase: FirebaseService,
    private router: Router, private usuario: UsuarioService
  ) {
    this.initializeApp();
    this.platform = platform;
    this.usuario.getUser().subscribe(user => {
      this.email = user.email;
      this.foto = user.foto;
    });
  }

  initializeApp() {
    this.platform.ready().then(() => {
      this.statusBar.styleDefault();
      this.splashScreen.hide();
      this.firebase.auth().onAuthStateChanged( user => {
        if (!user) {
          this.router.navigate(['login']);
        } else {
          this.router.navigate(['home']);
        }
      })
    });
  }
}
```

Fonte: Autoria própria (2018)

Figura 30 – Código Ionic – Tela de alarme HTML

```
<ion-header *ngIf="!email==''">
  <ion-toolbar color="primary">
    <ion-buttons slot="start">  <ion-menu-button></ion-menu-button>  </ion-buttons>
    <ion-title> Alarme </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content padding *ngIf="!email==''">
    <ion-card color="light">
                <h4 text-center>Central de alarme</h4>
       <ion-img src="../../assets/imagem2.jpg"></ion-img>
        <ion-card-content text-center>
            <ion-button size="medium" shape="round" fill="outline" expand="block" color="success" (click)="acao('ativar')">
              <ion-icon name="lock" slot="start"></ion-icon>
              Ativar
            </ion-button> <br>
            <ion-button size="medium" shape="round" fill="outline" expand="block" color="tertiary" (click)="acao('desativar')">
              <ion-icon name="unlock" slot="start"></ion-icon>
              Desativar
            </ion-button> <br> <br>
            <ion-button size="medium" shape="round" fill="outline" expand="block" color="danger" (click)="acao('disparar')">
              <ion-icon name="megaphone" slot="start"></ion-icon>
              Disparar
            </ion-button>
            <ion-item>
              <ion-icon name="warning" color="danger"></ion-icon>
              <ion-item>
                <h6 >PRESSIONE SOMENTE EM CASOS DE EMERGÊNCIA</h6>
              </ion-item>
            </ion-item>
        </ion-card-content>
    </ion-card>
</ion-content>
```

Fonte: Autoria própria (2018)

Figura 32 – Código Ionic – Tela de alarme TypeScript – Parte 1

```
import { Component, OnInit } from '@angular/core';
import { Http, Headers } from '@angular/http';
import { ActionSheetController } from '@ionic/angular';
import { FirebaseService } from '../services/firebase.service';
import { UsuarioService } from '../services/usuario.service';

@Component({
  selector: 'app-alarme',
  templateUrl: './alarme.page.html',
  styleUrls: ['./alarme.page.scss'],
})
export class AlarmePage implements OnInit {

  public email;
  constructor(private http: Http, public actionSheetController: ActionSheetController,
              private firebase: FirebaseService, private usuario: UsuarioService){
    this.usuario.getUser().subscribe(user => {
      this.email = user.email;
    });
  }

  ngOnInit() {
  }

```

Fonte: Autoria própria (2018)

Figura 33 – Código Ionic – Tela de alarme TypeScript – Parte 2

```
25      async acao(parametro) {
26        const actionSheet = await this.actionSheetController.create({
27          header: 'Tem certeza que deseja ' + parametro + ' alarme?',
28          buttons: [{
29            text: 'Confirmar', icon: 'checkmark-circle',
30            handler: () => {
31              console.log('Confirmado');
32              this.solicitar(parametro);
33            }
34          }
35          , {
36            text: 'Cancelar', icon: 'close', role: 'cancel',
37            handler: () => {
38              console.log('Cancelado');
39            }
40          }]
41        });
42        await actionSheet.present();
43      }
44
45      async solicitar(parametro){
46        this.http.get('http://192.168.15.110/' + parametro, {}).subscribe(
47        sucesso => { console.log(sucesso.json()) },
48        erro => {console.error(erro) }
49        );
50      }
51    }
```

Fonte: Autoria própria (2018)

Figura 35 – Código Ionic – Tela de Luzes HTML

```
1   <ion-header *ngIf="!email==''">
2     <ion-toolbar color="primary">
3       <ion-buttons slot="start">
4         <ion-menu-button></ion-menu-button>
5       </ion-buttons>
6       <ion-title>
7         Luzes
8       </ion-title>
9     </ion-toolbar>
10  </ion-header>
11
12  <ion-content padding *ngIf="!email==''">
13      <ion-card color="light">
14        <h4 text-center>Luz da sala</h4>
15        <ion-img src="../../assets/imagem3.jpg"></ion-img>
16        <ion-card-content text-center>
17          <ion-button size="medium" shape="round" fill="outline" expand="block" color="success" (click)="acao('acender')">
18            <ion-icon name="bulb" slot="start"></ion-icon>
19            Acender
20          </ion-button>
21          <br>
22          <ion-button size="medium" shape="round" fill="outline" expand="block" color="danger" (click)="acao('apagar')">
23            <ion-icon name="moon" slot="start"></ion-icon>
24            Apagar
25          </ion-button>
26        </ion-card-content>
27      </ion-card>
28  </ion-content>
```

Fonte: Autoria própria (2018)

Figura 37 – Código Ionic –Tela de Portas HTML

```
<ion-header *ngIf="!email==''">
  <ion-toolbar color="primary">
    <ion-buttons slot="start">
      <ion-menu-button></ion-menu-button>
    </ion-buttons>
    <ion-title>
      Portas
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content padding *ngIf="!email==''">
    <ion-card color="light">
      <h4 text-center>Porta de entrada</h4>
      <ion-img src="../../assets/imagem4.jpg"></ion-img>
      <ion-card-content text-center>
        <ion-button size="medium" shape="round" fill="outline" expand="block" color="success" (click)="acao('abrir')">
          <ion-icon name="trending-up" slot="start"></ion-icon>
          Abrir
        </ion-button>
        <br>
        <ion-button size="medium" shape="round" fill="outline" expand="block" color="danger" (click)="acao('fechar')">
          <ion-icon name="undo" slot="start"></ion-icon>
          Fechar
        </ion-button>
      </ion-card-content>
    </ion-card>
</ion-content>
```

Fonte: Autoria própria (2018)

Figura 39 – Código Ionic –Tela Sobre HTML

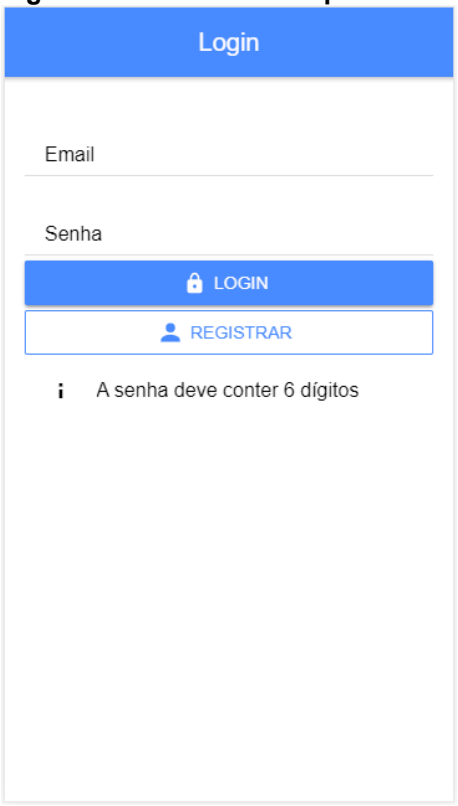
```
<ion-header *ngIf="!email==''">
  <ion-toolbar color="primary">
    <ion-buttons slot="start">  <ion-menu-button></ion-menu-button>  </ion-buttons>
    <ion-title> Contato </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content *ngIf="!email==''">
  <ion-card color="light">
    <ion-list>
      <ion-card-header>
        <h3 text-center>Entre em contato com os desenvolvedores:</h3>
      </ion-card-header>
      <br><br>
      <ion-item>
        <ion-avatar slot="start">
          <ion-img src="../../assets/bruno.jpg"> </ion-img>
        </ion-avatar>
        <ion-label>
          <h2>Bruno Vendrametto</h2>
          <p>brvendrametto@gmail.com</p>
        </ion-label>
        <ion-buttons end>
          <ion-button href="https://www.facebook.com/bruno.vendrametto" fill=""><ion-icon name="logo-facebook"></ion-icon></ion-button>
          <ion-button href="https://www.linkedin.com/in/brvendrametto/" fill=""><ion-icon name="logo-linkedin"></ion-icon></ion-button>
        </ion-buttons>
      </ion-item>
      <br>
      <br>
    </ion-list>
  </ion-card>
</ion-content>
```

Fonte: Autoria própria (2018)

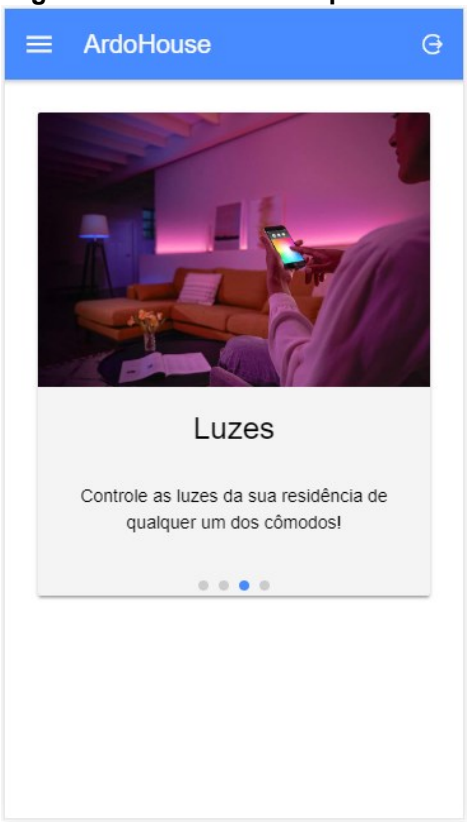
APÊNDICE C - Interface do aplicativo móvel

Figura 19 – Interface do aplicativo – Página de Login



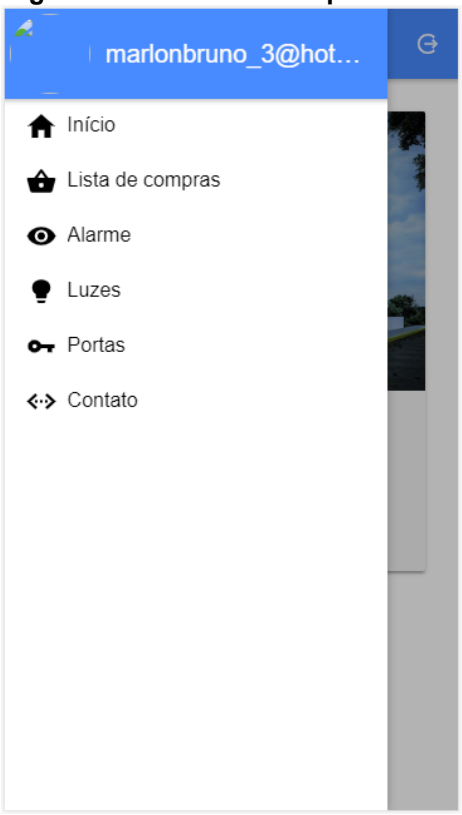
Fonte: Autoria própria (2018)

Figura 21 – Interface do aplicativo – Página Home



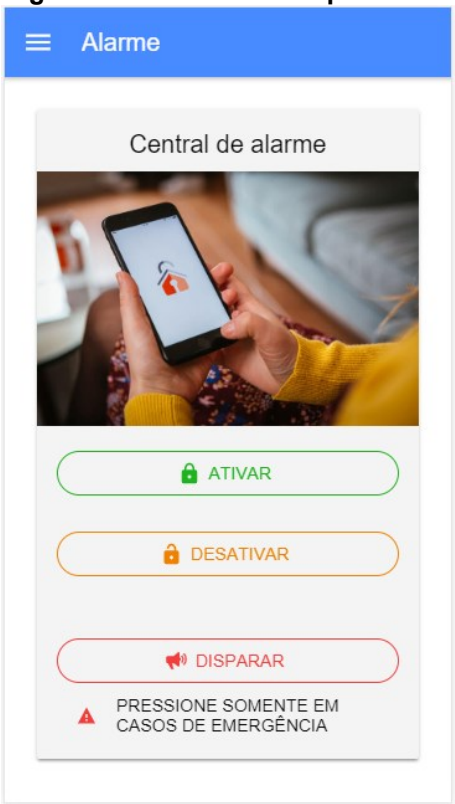
Fonte: Autoria própria (2018)

Figura 24 – Interface do aplicativo – Menu lateral



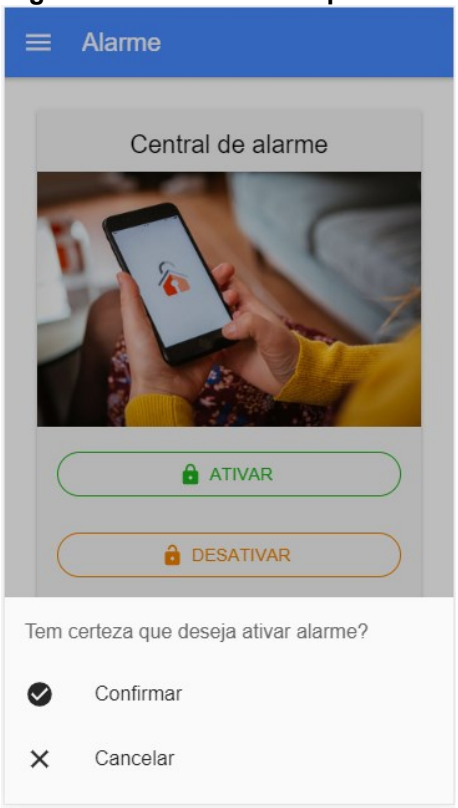
Fonte: Autoria própria (2018)

Figura 29 – Interface do aplicativo – Tela de Alarme



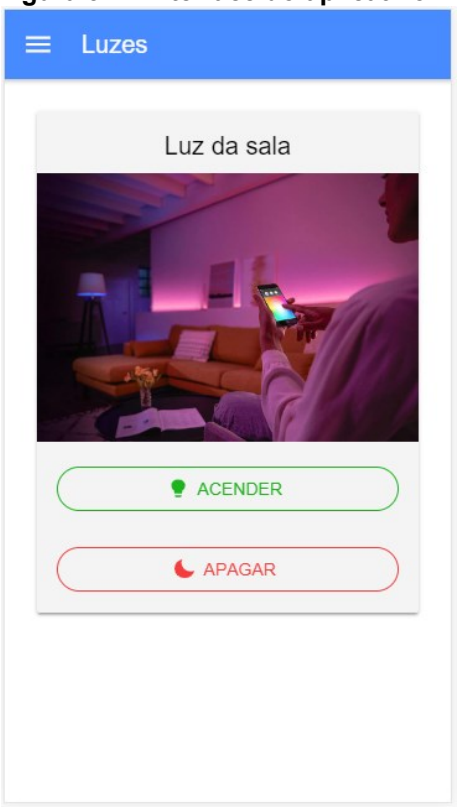
Fonte: Autoria própria (2018)

Figura 31 – Interface do aplicativo – Confirmação de ação



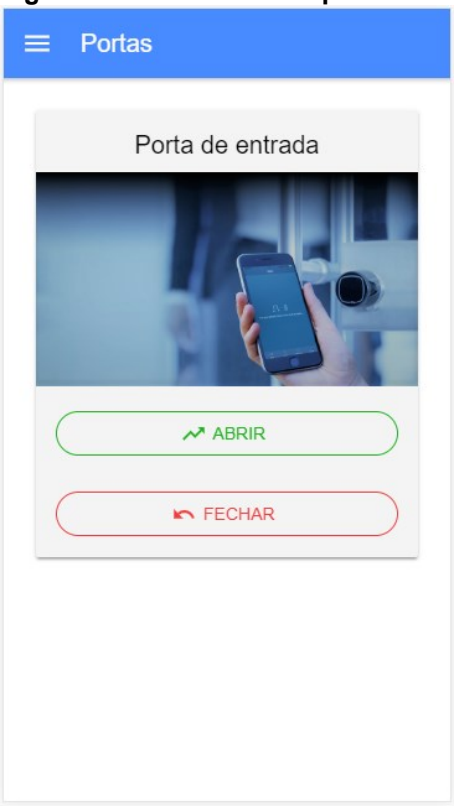
Fonte: Autoria própria (2018)

Figura 34 – Interface do aplicativo – Tela de Luzes



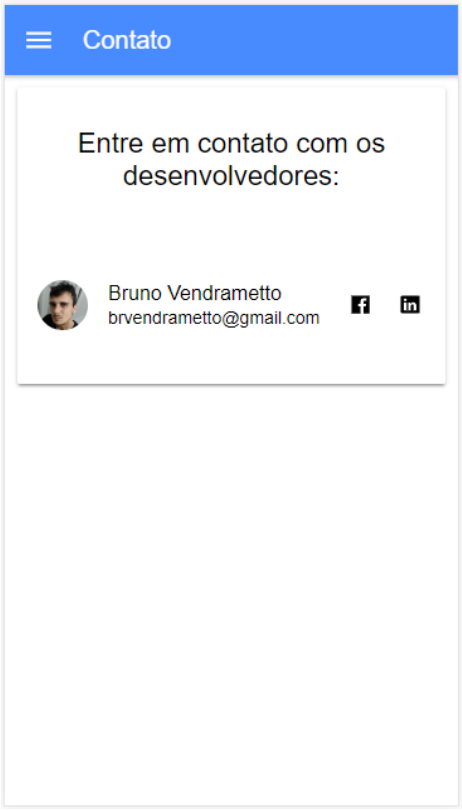
Fonte: Autoria própria (2018)

Figura 36 – Interface do aplicativo – Tela de Portas



Fonte: Autoria própria (2018)

Figura 38 – Interface do aplicativo – Tela de Contato



Fonte: Autoria própria (2018)