UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

FELIPE AGUIAR DE CARVALHO

Controle *ON/OFF* de equipamentos residenciais com o *kit* de desenvolvimento *Particle Core*

São Carlos
2015

FELIPE AGUIAR DE CARVALHO

CONTROLE *ON/OFF* DE EQUIPAMENTOS RESIDENCIAIS COM O *KIT* DE DESENVOLVIMENTO *PARTICLE CORE*

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com Ênfase em Eletrônica

ORIENTADOR: Prof. Edson Gesualdo

São Carlos
2015

Carvalho, Felipe Aguiar de
C331c Controle ON/OFF de equipamentos residenciais com o kit de desenvolvimento Particle Core / Felipe Aguiar de Carvalho; orientador Edson Gesualdo. São Carlos, 2015.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2015.

1. Automação residencial. 2. Internet das Coisas. 3. Particle Core. 4. Android. I. Título.

FOLHA DE APROVAÇÃO

Nome: Felipe Aguiar de Carvalho

Título: "Controle ON/OFF de equipamentos residenciais com o kit de desenvolvimento Particle Core"

Trabalho de Conclusão de Curso defendido e aprovado em 01 / 12 / 2015,

com NOTA 9,3 (nove, três), **pela Comissão Julgadora:**

Prof. Assistente Edson Gesualdo - (Orientador - SEL/EESC/USP)

Prof. Associado Evandro Luis Linhari Rodrigues - (SEL/EESC/USP)

Mestre Heitor Vinicius Mercaldi - (Doutorando - SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Dr. José Carlos de Melo Vieira Júnior

AGRADECIMENTOS

Ao professor Edson Gesualdo pelo apoio oferecido durante toda a graduação.

A nova regra para o futuro será “qualquer coisa que pode ser conectada, estará conectada”. [Jacob Morgan]

RESUMO

O cenário atual da área de automação residencial aponta para a necessidade da utilização de um sistema operacional de código aberto que incentive o investimento na integração entre diferentes soluções e permita a criação de produtos que possam ser facilmente instalados, utilizados e atualizados. A empresa estadunidense *Particle* oferece uma solução inovadora de código aberto para realizar o controle de dispositivos conectados à nuvem através de um pacote de ferramentas de *software* e *hardware* que utilizam o conceito da "*Internet* das Coisas". No presente trabalho, os produtos da *Particle* foram utilizados no âmbito da domótica para realizar o acionamento de equipamentos residenciais pela *internet* através de um aplicativo para dispositivos móveis com o sistema operacional *Android*. O trabalho foi testado com dois *kits* de desenvolvimento *Wi-Fi* oferecidos pela empresa, o *Particle Core* e o *Particle Photon*, apresentando boa resposta para ambos. Na automação residencial, diferentemente da automação industrial, respostas rápidas, precisas e altamente imunes a falhas podem dar lugar ao conceito de produtos fáceis de operar e com um bom acabamento. Pelo tamanho compacto, o *kit* de desenvolvimento utilizado permite a concepção de produtos de fácil integração em ambientes domésticos. Além disso, a *interface* do aplicativo desenvolvido nesse trabalho foi projetada para atender a demanda do mercado da domótica, utilizando elementos intuitivos para usuários de dispositivos móveis. Através do aplicativo, é possível gerenciar diversos dispositivos da *Particle*, controlando os equipamentos conectados às suas saídas e os organizando por ambientes da residência. O projeto apresenta, também, uma alta capacidade de adaptação para diferentes aplicações, como o controle dos mais diversos tipos de equipamentos.

Palavras-chave: Automação residencial. Internet das Coisas. *Particle Core*. *Android*.

ABSTRACT

The current scenario of home automation points to the need for an open source operating system that encourages investment in the integration of different solutions and allows the concept of products that are easy to install, to use and to update. The American company Particle offers an innovative open source solution for controlling devices connected to the cloud through a package of software and hardware tools that use the concept of the “Internet of Things”. In this study, within the home automation scope, the Particle products performed the on-off control of residential equipment over the internet through an Android application for mobile devices. The work was tested with two Wi-Fi development kits offered by the company, the Particle Core and the Particle Photon, obtaining a good response to both. In home automation, different from industrial automation, highly accurate, fast and foolproof systems can give place to well designed and easy-to-use products. Because of its compact size, the development kit allows the design of products that are easy to integrate at home. In addition, the application interface design developed in this work meet the market demand of home automation, using intuitive elements to mobile users. Through the application, the user can manage many Particle devices, controlling residential equipment connected to its outputs and organizing them by rooms. The project is also highly adaptable to different applications such as the control of a wide range of equipment.

Keywords: Home automation. Internet of Things. Particle Core. Android.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

IoT	*Internet of Things*
API	*Application Programming Interface*
USB	*Universal Serial Bus*
CLP	Controlador Lógico Programável
PLC	*Power Line Communication*
IV	Infravermelho
RF	Radiofrequência
RAM	*Random Access Memory*
REST	*Representional State Transfer*
URL	*Uniform Resource Locator*
TLS	*Transport Layer Security*
LED	*Light Emitting Diode*
SDK	*Software Development Kit*
XML	*eXtensible Markup Language*
HTTP	*Hypertext Transfer Protocol*
PWM	*Pulse Width Modulation*
CDC	*Communications Device Class*
USART	*Universal Synchronous Asynchronous Receiver Transmitter*
SPI	*Serial Peripheral Interface*
I^2C	*Inter-Integrated Circuit*
CPU	*Central Processing Unit*
AURESIDE	Associação Brasileira de Automação Residencial
USP	Universidade de São Paulo
DCU	*Dublin City University*
EESC	Escola de Engenharia de São Carlos

SUMÁRIO

1. Introdução

1.1. Objetivos do trabalho

O objetivo desse trabalho é desenvolver um sistema de automação residencial capaz de realizar o controle *ON/OFF* de equipamentos através de um aplicativo para *Android*. Como exemplos de equipamentos que podem ser acionados, podem-se citar lâmpadas, tomadas, campainhas, regadores automáticos e portões eletrônicos.

Tais equipamentos devem ser conectados à *internet* através de um *kit* de desenvolvimento *Wi-Fi* chamado *Particle Core*. Utilizando o conceito da IoT (explicado com mais detalhes na seção "2.2. A *internet* das coisas"), o aplicativo desenvolvido deve se comunicar com as saídas digitais do *kit* e pode operar em qualquer dispositivo móvel com *Android* 5.0 ou superior com acesso à *internet*.

De forma intuitiva, o usuário pode interagir com o aplicativo para cadastrar os *kits* a que tem acesso e personalizar o aplicativo com o nome dos cômodos da residência e dos equipamentos ligados às saídas do *kit*. Na *interface* principal do aplicativo é exibida uma lista com os ambientes cadastrados pelo usuário. Ao selecionar um ambiente, uma nova tela é aberta apresentando os equipamentos cadastrados para aquele ambiente. O toque em um dos itens dessa tela deve enviar um requerimento para o *kit* correspondente, alterando o valor das suas saídas digitais e atualizando o *display* com recursos visuais que representem o novo valor.

O trabalho envolveu o desenvolvimento de um *firmware* para o *Particle Core*, bem como de um aplicativo para *Android* que fossem capazes de se comunicar através da *Particle Cloud*[1]. Ainda, um circuito foi projetado para receber os comandos do *kit* e controlar os aparelhos residenciais.

1.2. Divisão do trabalho

Esse trabalho está dividido em 6 partes:

[1] A *Particle Cloud* é um servidor de código aberto que oferece uma API para interagir com dispositivos que utilizem o protocolo de comunicação da *Particle*.

Em “2. O cenário”, são introduzidas informações atualizadas a respeito da área de automação residencial e de como o conceito da “*Internet* das Coisas” está sendo fortemente inserido nas discussões sobre tecnologia nos últimos anos. Essas informações foram fundamentais para a determinação de como esse trabalho seria desenvolvido.

Em “3. A *Particle*”, citam-se informações relevantes sobre os produtos utilizados e sobre o surgimento da empresa *Particle*, que visa ser uma facilitadora de projetos como o descrito nesse trabalho. A escolha desses produtos foi fortemente influenciada pelas informações evidenciadas no capítulo “2. O cenário”.

As informações contidas nos capítulos 4, 5 e 6 apresentam os fundamentos e metodologias empregados no desenvolvimento desse trabalho. O teste do sistema desenvolvido juntamente com as imagens resultantes da interface com o usuário é apresentado na seção “6.3. Funcionamento do aplicativo”. Essa divisão visa facilitar a explicação do projeto como um todo.

Em “4. O circuito de acionamento”, são descritos os circuitos utilizados para testar o funcionamento do projeto e como eles podem ser alterados ou adaptados para controlar outros equipamentos.

Em “5. O *firmware*”, são dadas informações a respeito do *firmware* desenvolvido para rodar no *kit* de desenvolvimento, que deve atender às requisições provenientes da *Particle Cloud*. O Apêndice C contém todo o código utilizado no desenvolvimento do *firmware*.

Em “6. O aplicativo para Android”, é detalhado o funcionamento do aplicativo desenvolvido nesse trabalho, considerando que o leitor possua familiaridade com alguma linguagem orientada a objetos. A ideia do capítulo é, entretanto, abstrair ao máximo os conceitos técnicos envolvidos na linguagem e prover ao leitor um entendimento consistente do ciclo de operação da aplicação. Caso seja de interesse do leitor, o código-fonte do aplicativo desenvolvido pode ser encontrado no endereço “https://github.com/acfelipe/my-app” (CONTROLE [...], 2015).

Em “7. Conclusões”, são apresentadas notas a respeito do *kit* de desenvolvimento utilizado, as fragilidades do sistema desenvolvido e como elas podem ser solucionadas no futuro, o motivo do aplicativo para *Android* ter sido desenvolvido da forma em que foi e as disciplinas do curso “Engenharia Elétrica – Ênfase em Eletrônica” que auxiliaram no desenvolvimento desse trabalho.

O contato para troca de informações, sugestões ou dúvidas a respeito do projeto pode ser feito através do e-mail “felipe_a.c@hotmail.com”.

2. O cenário

2.1. A automação residencial

A automação residencial, também chamada de domótica, se refere ao uso de sistemas para controlar o funcionamento de equipamentos residenciais de forma automatizada e, em alguns casos, remotamente. A área é originária da automação industrial, que teve nos dispositivos CLPs (Controladores Lógicos Programáveis) uma grande revolução graças aos avanços da microeletrônica (BORTOLUZZI, 2013).

Os dois mercados, entretanto, apresentam diferentes demandas: enquanto na automação industrial é fundamental que os equipamentos operem com respostas rápidas e precisas e uma alta imunidade a falhas, na automação residencial tais características têm menos importância e podem dar lugar a produtos com um melhor acabamento e com *interfaces* amigáveis e intuitivas.

O marco inicial da domótica data da década de 70, quando surgiu nos Estados Unidos o protocolo X10, que utilizava a tecnologia PLC (*Power Line Communication*) para comunicação entre dispositivos eletrônicos para automação residencial. O protocolo utilizava a linha de alimentação doméstica para o controle dos dispositivos através de sinais na rede elétrica que representavam informações digitais. Tal tipo de comunicação é, no entanto, altamente suscetível a interferência elétrica (ROTHFELD, 2015).

Com o passar do tempo, outras tecnologias foram incorporadas à automação doméstica, como controles remotos programáveis com comunicação por radiação infravermelha (IV) ou radiofrequência (RF). Mais à frente, a *internet* de banda larga e o crescente uso de dispositivos móveis deu origem a uma nova forma de controle e monitoramento de residências.

Hoje, a automação residencial pode incluir uma vasta gama de funcionalidades. Alguns exemplos são o agendamento de operações automatizadas de irrigação de jardins, o controle de temperatura de sistemas de condicionadores de ar ou aquecedores, a dimerização da iluminação de ambientes, o acionamento automático de cortinas e portões e o acesso remoto a informações da residência, como o sistema de câmeras de segurança.

No Brasil, “o mercado de automação residencial já fatura R$500 milhões [ao ano] com um crescimento anual de 30%” (TECHINBRAZIL, 2015). Segundo Muratori (2013), diretor executivo da Associação Brasileira de Automação Residencial (AURESIDE), “o número de fornecedores

[de equipamentos de automação residencial] triplicou em menos de cinco anos e embora se trate de um crescimento considerável, ainda vivemos a infância desse mercado".

2.2. A *Internet* das Coisas

A "*Internet* das Coisas" (IoT) é a rede de objetos físicos que contém tecnologia embarcada para se comunicar com a *internet* e sensores para interagir com o ambiente externo. A ideia por trás da IoT está em conectar basicamente qualquer dispositivo com uma chave *ON/OFF* à *internet*, desde celulares, cafeteiras, máquinas de lavar, fones de ouvido ou lâmpadas até componentes internos de máquinas, como o motor a jato de um avião ou a broca de uma plataforma de petróleo (MORGAN, 2014).

Hoje em dia, a *internet* de banda larga está disponível para mais pessoas ao redor do mundo, mais dispositivos com sensores internos e conectividade *Wi-Fi* estão sendo criados, o custo das tecnologias envolvidas está caindo e a popularidade dos dispositivos móveis está em rápida ascensão. Todos esses fatores combinados estão criando o cenário perfeito para a explosão da IoT (MORGAN, 2014).

A empresa de consultoria *Gartner* (2013) afirma que em 2020 haverá, excluindo os computadores, smartphones e tablets, 26 bilhões de dispositivos conectados à *internet*, 30 vezes mais comparado a 2009. A nova regra para o futuro será "qualquer coisa que pode ser conectada, estará conectada" (MORGAN, 2014).

2.3. O futuro das empresas de automação residencial

O atual cenário da automação residencial aponta para o surgimento de empresas focadas no desenvolvimento de produtos elegantes, fáceis de instalar e utilizar e altamente integráveis com outros equipamentos.

De acordo com Wortmeyer, Freitas e Cardoso (2005), "A falta de integração entre os diversos sistemas, devido ao fato de muitos produtos serem lançados de forma isolada, contribui para a dificuldade de se operar todos esses equipamentos separadamente".

Segundo a *TechinBrazil* (2015), “a criação de um sistema operacional de código aberto para a tecnologia seria essencial para facilitar a oferta e instalação das soluções, assim como incentivar o investimento na integração, permitindo que negócios menores entrem no mercado”.

Esse trabalho está focado exatamente nessa premissa: utilizar um sistema de código aberto altamente integrável para o desenvolvimento de um produto que possa ser facilmente instalado, utilizado e atualizado.

Atualmente as empresas de automação residencial desprendem muito esforço no desenvolvimento da infraestrutura básica por trás dos seus produtos. A ideia dos produtos da *Particle* utilizados nesse trabalho é mudar esse cenário, oferecendo toda a infraestrutura necessária para que novas empresas que utilizam o conceito da IoT possam surgir no mercado.

3. A *Particle*

3.1. Introdução

A *Particle*, anteriormente chamada *Spark*, é uma *startup*[1] fundada em 2012 em São Francisco, Califórnia. A ideia da empresa é oferecer um pacote de ferramentas de *software* e *hardware* para ajudar outras empresas a prototiparem, escalarem e gerenciarem seus produtos de IoT, sendo assim uma solução completa e de código aberto para dispositivos conectados à nuvem.

No *github*[2] da *Particle*, disponibilizado através do endereço "http://spark.github.io" (PARTICLE SOURCE, 2015), é possível encontrar toda a documentação necessária para construir do zero um produto conectado à *internet*, desde *designs* de *hardware*, *firmware* e *cloud software* (servidor em nuvem) até modelos de aplicativos e ferramentas de desenvolvimento.

No dia primeiro de junho de 2013, uma campanha de financiamento coletivo no *Kickstarter*[3] conseguiu arrecadar $567.968, contando com 5.549 apoiadores para viabilizar o projeto do *Particle Core*, o *kit* de desenvolvimento *Wi-Fi* que foi utilizado nesse trabalho. Mais à frente, no dia 12 de novembro de 2014, foi anunciado o *Particle Photon*, uma versão melhorada do *Particle Core* que foi lançada de fato em março de 2015. Esse trabalho foi testado com ambas as versões do produto, operando com um tempo de atraso adequado para aplicações residenciais. Por simplificação, referir-se-á sempre aos *kits* como *Particle Core* (ou apenas *Core*), uma vez que o trabalho começou a ser desenvolvido utilizando a primeira versão do *kit*. Essa versão, no entanto, não se encontra mais disponível para compra.

É bastante comum a obsolescência juvenil de produtos em empresas *startups*. Isso se deve ao fato de que esses produtos normalmente estão em fase de validação da ideia, o que significa que as demandas do mercado não são absolutamente claras. Dessa forma, essas empresas utilizam um processo de inovação contínua, aprimorando seus produtos através de constantes *feedbacks* da comunidade de usuários.

---

[1] *Startups* são empresas em fase de desenvolvimento que possuem um modelo de negócios escalável e buscam soluções inovadoras em condições de extrema incerteza.
[2] *Github* é um serviço de *Web Hosting* comumente utilizado para compartilhamento de código e desenvolvimento de *software*.
[3] *Kickstarter* é um *website* de financiamento coletivo que busca apoiar projetos inovadores.

Atualmente, a *Particle* oferece dois produtos principais: o *Particle Photon* (por $19), para criação de produtos genéricos conectados à *internet* e o *Particle Electron* (por $39), para criação de celulares ou produtos que se comuniquem através de sinais provenientes de torres de telefonia celular. O segundo, no entanto, ainda está em fase de desenvolvimento, com data de lançamento prevista para janeiro de 2016.

A ideia em utilizar os produtos da *Particle* é aproveitar toda a infraestrutura oferecida pela empresa para a criação de produtos conectados à *internet*, como seus *kits* de desenvolvimento compactos (o *Particle Core* possui apenas 3,7cm de comprimento e 2cm de largura) projetados especialmente para aplicações que envolvam conexão com a nuvem, sua documentação detalhada de API, seus *templates* de *software* e ferramentas de desenvolvimento disponibilizados gratuitamente e seu fórum *online* para discussão de problemas, soluções de dúvidas e integração com a comunidade de desenvolvedores.

Mais informações sobre a *Particle* podem ser encontradas em sua página oficial, “https://www.particle.io/” (PARTICLE, 2015).

3.2. O *Particle Core*

O *Particle Core* é um *kit* de desenvolvimento *Wi-Fi* para *hardwares* conectados à *internet*. Ele possui um microcontrolador *STM32F103 72MHz ARM Cortex M3* que roda uma única aplicação – ou *firmware* –, um módulo *Wi-Fi Texas Instruments CC3000*, além de 128kB de *flash*, 20kB de RAM e 2MB de *flash* externa.

O *kit* pode ser alimentado por uma fonte não regulada com tensão entre 3,6V e 6V ou através de uma entrada micro USB que forneça alimentação de 5V. Quando conectado via USB, o pino V_{IN}, utilizado para alimentação através da fonte, fornece tensão de 5V, que pode ser usada para alimentar componentes externos.

No Apêndice A, podem ser encontradas informações técnicas a respeito do *Particle Core*, bem como um esquema ilustrando a pinagem do *kit*. No Apêndice B, explica-se o caminho para conectar o *kit* na *internet* pela primeira vez.

O *Particle Core*, quando conectado à *internet*, estabelece uma conexão com a *Particle Cloud*. Ao se conectar com essa nuvem, o *Core* se torna acessível de qualquer lugar através de

uma API com arquitetura REST (*Representional State Transfer)*, projetada para facilitar a troca de dados com o *Core* através da *internet*.

Cada *Particle Core* tem uma URL (*Uniform Resource Locator)* própria. Através dessa URL, aplicativos na *web* podem enviar requerimentos ou receber informações. Nesse trabalho, os requerimentos devem acionar funções internas do *Core* alterando o valor das suas saídas digitais. A saídas, por sua vez, são conectadas a módulos relés que devem ativar os circuitos de alta potência da rede doméstica.

A conexão entre o aplicativo e o *Core* através da *Particle Cloud* é feita de forma privada utilizando o protocolo de segurança TLS (*Transport Layer Security*). Uma chave de segurança denominada *Access Token* é enviada ao *Core* junto com todos os requerimentos.

Como mencionado acima, esse trabalho foi desenvolvido utilizando o servidor da *Particle Cloud* para realizar a comunicação com o *Core*. O controle e a segurança das informações trocadas com os dispositivos da *Particle*, no entanto, ficam dependentes dos mecanismos envolvidos na comunicação com esse servidor. Em um projeto mais sofisticado ou na possível adaptação desse trabalho para um produto comercial, é indicado o desenvolvimento de um servidor próprio. Para referência na criação de um servidor próprio, o *design* da *Particle Cloud* é disponibilizado através do endereço “http://spark.github.io” (PARTICLE SOURCE, 2015).

4. O circuito de acionamento

Para a construção do circuito de acionamento, são necessários:

- Um *plug* macho de tomada a ser conectado na rede elétrica de 110 ou 220V (dependendo da tensão de operação dos equipamentos residenciais que se deseja controlar) para alimentação do sistema;
- Um adaptador de energia USB para redes de 110 ou 220V (dependendo da rede em que o sistema for conectado) com saída igual a 5V;
- Um cabo USB *AM/Micro*;
- Um ou mais *kits* de desenvolvimento *Particle Core*;
- Um ou mais módulos relés com tensão de operação de 5V;
- Um ou mais equipamentos residenciais que se deseja controlar;
- Cabos e fios para a conexão dos componentes do sistema.

A Figura 1 apresenta o esquema de um circuito de acionamento básico para controle de uma lâmpada comum com *spot*.

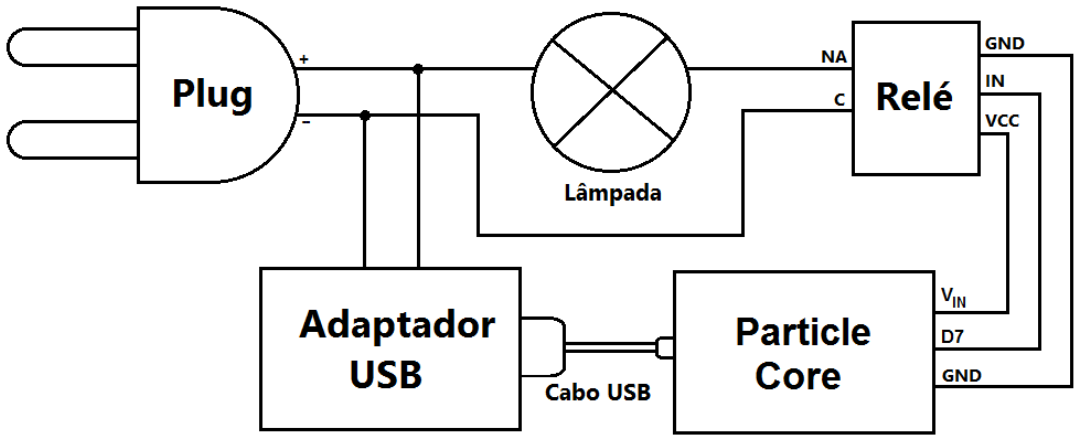


Figura 1 - Circuito para acionamento de uma lâmpada

Nesse esquema, o *kit* de desenvolvimento é alimentado através da conexão USB, liberando o pino V_{IN} do *Core* para ser utilizado como alimentação de 5V para o módulo relé. Na Figura 2 é apresentada montagem do circuito na prática.

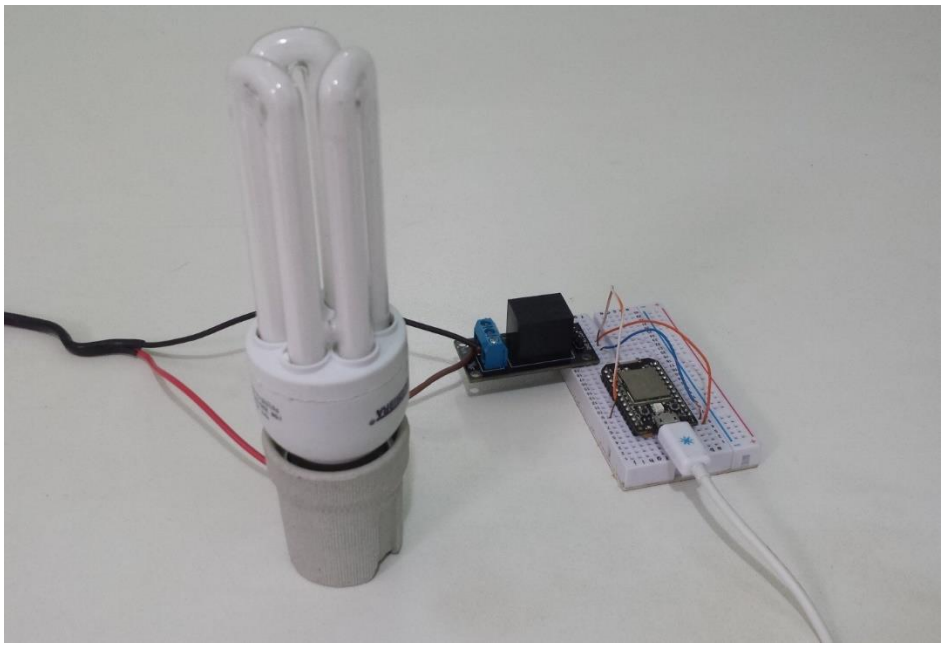


Figura 2 - Montagem do circuito básico

Para testar o sistema, foi utilizada a saída D7 como acionadora do relé, uma vez que ela é conectada ao LED do *kit*, permitindo acompanhar o funcionamento do circuito.

Esse circuito pode ser ampliado e adaptado para outras funcionalidades. A Figura 3 mostra um exemplo de circuito utilizando diversos *Cores* e equipamentos. Esse esquema foi utilizado para testar o funcionamento do aplicativo, descrito na seção “6.3. Funcionamento do aplicativo”.

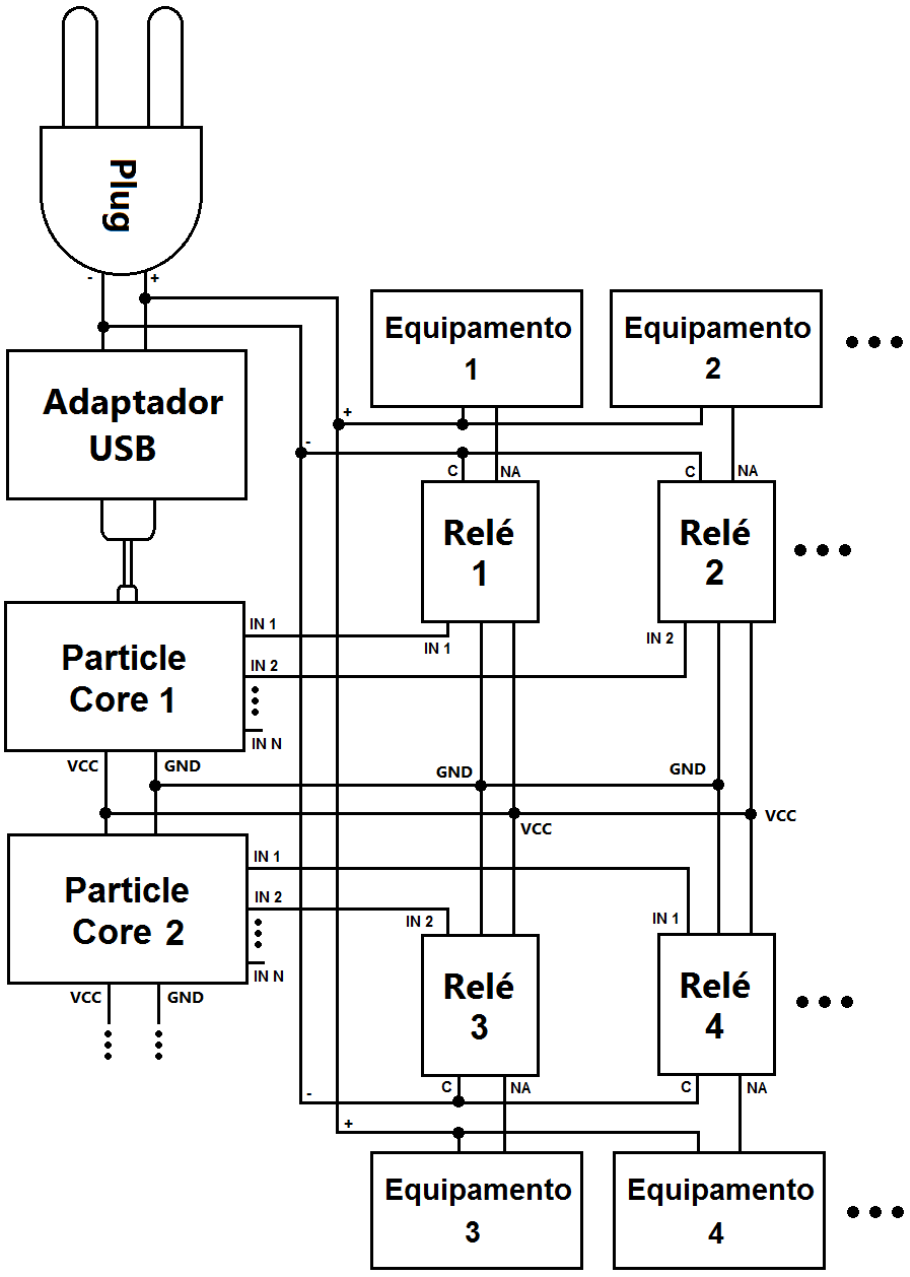


Figura 3 - Circuito com múltiplos *Cores* e equipamentos

5. O *firmware*

5.1. Introdução

Chamaremos de *firmware* a aplicação que será executada no microcontrolador do *Particle Core*. Sua função é receber requerimentos provenientes da *Particle Cloud* e responder a eles com as informações ou com os comandos requisitados.

Para que essa comunicação seja feita instantaneamente, o *Core* possui instalado o sistema operacional de tempo real *FreeRTOS*. Esse tipo de sistema operacional é projetado para atender requisições em tempo real, com o tempo de processamento medido em frações de segundos. O *FreeRTOS* é um sistema operacional bastante robusto e livre para utilização em produtos comerciais. Mais informações podem ser encontradas no endereço oficial do sistema, “http://www.freertos.org/” (FREERTOS, 2015).

Nesse trabalho, o *firmware* foi desenvolvido para responder a apenas um tipo de requerimento: o de escrita em uma das saídas digitais. O desenvolvimento do *firmware* para o *Core* foi feito utilizando a plataforma *online* oferecida pela *Particle* no endereço “http://build.particle.io” (PARTICLE BUILD, 2015).

A linguagem utilizada no desenvolvimento de *firmwares* para dispositivos da *Particle* é baseada no *framework*[1] de programação para microcontroladores *Wiring*. O *Wiring* é uma linguagem de código aberto que permite a programação de dispositivos conectados em uma grande gama de diferentes microcontroladores. A documentação detalhada da linguagem pode ser encontrada no endereço “http://wiring.org.co/reference/” (WIRING REFERENCE, 2015).

5.2. Desenvolvimento do firmware

Todo programa baseado em *Wiring* possui duas funções essenciais:

- *setup()*: roda uma única vez no início da aplicação, quando o dispositivo inicializa ou reinicializa;

[1] Um *framework* é um conjunto de códigos utilizados para auxiliar o desenvolvimento de *software*, provendo funcionalidades genéricas para aplicações.

- *loop()*: roda continuamente durante o funcionamento da aplicação.

No início do *firmware* foram declaradas quatro funções que são capazes de ler/escrever valores digitais ou analógicos nas entradas/saídas do *Core*. A única função que é de fato utilizada para o controle dos equipamentos residenciais descrito nesse trabalho é a *myAppDigitalWrite()*, utilizada para alterar o valor das saídas digitais do *Core*.

Em seguida, dento da função *setup()*, foram chamadas as funções *Spark.function()* para definir os nomes que são utilizados para referenciar essas funções através da *Particle Cloud*.

A função *loop()* foi declarada em seguida. Porém, não é necessário colocar nenhum código dentro dela, uma vez que nenhuma operação estática deve ser realizada enquanto não houver requisições da *Particle Cloud*.

As funções declaradas no início do *firmware* devem receber uma *String* (sequência de caracteres) como parâmetro. No caso da função *myAppDigitalWrite()* mencionada anteriormente, a *String* deve ser decodificada para identificar qual saída digital (D0 a D7) deve ser acionada e qual valor (alto ou baixo) deve ser aplicado à ela.

Caso a operação seja realizada com sucesso, a função retorna o número inteiro um. Do contrário, a função retorna um valor negativo que corresponde ao tipo de erro que pode ter ocorrido.

No Apêndice C pode ser encontrado o código completo e comentado utilizado no desenvolvimento do *firmware*.

6. O Aplicativo para *Android*

6.1. Introdução

Para o desenvolvimento do aplicativo para *Android*, foi utilizado o *software Android Studio* 1.3.2, disponível para *download* no endereço “http://developer.android.com/sdk” (ANDROID STUDIO, 2015). A linguagem oficial para desenvolvimento de aplicativos para *Android* é o *Java*.

O *Java* é uma linguagem de programação interpretada orientada a objetos desenvolvida na década de 90. Diferentemente das linguagens convencionais, que são compiladas para o código nativo, a linguagem *Java* é compilada para um *bytecode* que é executado por uma máquina virtual.

Este capítulo descreverá as principais características do aplicativo para *Android* desenvolvido nesse trabalho, considerando que o leitor possua familiaridade com o funcionamento de uma linguagem orientada a objetos. A ideia do capítulo é, entretanto, abstrair ao máximo os conceitos técnicos envolvidos na linguagem e prover ao leitor um entendimento consistente do ciclo de operação da aplicação.

Todo o código do aplicativo *Android* desenvolvido nesse trabalho está disponível no endereço “https://github.com/acfelipe/my-app” (CONTROLE [...], 2015). Para executar o aplicativo, é necessário um dispositivo (ou emulador de dispositivo) que possua instalado o sistema operacional *Android* de versão 5.0 (chamada popularmente de *Lollipop*) ou superior com acesso à *internet* e 5MB de memória interna disponível para a sua instalação. Antes da utilização do aplicativo, deve ser realizada a reivindicação do *Core* descrita no Apêndice B.

Por se tratar de uma aplicação leve e com pouca demanda de *hardware* se comparado à capacidade de processamento dos dispositivos móveis modernos, não foi feito um estudo a respeito do consumo de recursos do dispositivo pela aplicação.

Também, a velocidade de comunicação com o *Core* é dependente de fatores externos, como a velocidade da conexão com a *internet* que o sinal *Wi-Fi* oferece. Por isso, não foi calculado o tempo envolvido na comunicação com o *Core*. Todavia, em um teste com *internet* de 30MB pôde-se notar um *delay* da ordem de décimos de segundos.

6.2. O *Particle Android* SDK

O *Particle Android* SDK é um *kit* de desenvolvimento de *software* que a *Particle* oferece para simplificar operações básicas no desenvolvimento de aplicativos para *Android* envolvendo a comunicação com dispositivos da *Particle* (incluindo o *Core* e o *Photon* utilizados nesse trabalho). A biblioteca da *Particle Android* SDK é dividida em duas partes: a *Particle Android Device Setup* e a *Particle Android Cloud* SDK. A primeira permite criar um processo de configuração inicial simples para usuários cadastrarem seus *Cores*. A segunda oferece métodos para que o aplicativo interaja com o *Core* através da *Particle Cloud*, permitindo, atualmente:

- Obter uma lista de dispositivos da *Particle* cadastrados por um usuário;
- Ler variáveis dos dispositivos;
- Chamar funções nos dispositivos;
- Gerenciar os *Access Tokens* dos dispositivos;
- Reivindicar dispositivos para a conta de um usuário.

No futuro, entretanto, será possível através da *Particle Android Cloud* SDK publicar eventos dos aplicativos *Android* para a *Particle Cloud* e se inscrever para eventos públicos ou privados publicados pelos dispositivos da *Particle* (PARTICLE REFERENCE, 2015).

O *Particle Android* SDK, entretanto, ainda está em fase *beta* de desenvolvimento. Apesar de ser testado e na maior parte das vezes estável, podem ocorrer *bugs* na sua utilização. Por essa razão, nesse trabalho decidiu-se por não aproveitar os benefícios provenientes das bibliotecas do *Particle Android* SDK.

6.3. Funcionamento do aplicativo

Para testar o funcionamento do aplicativo, foi simulado o sistema de uma casa com quatro cômodos automatizados (cozinha, garagem, quarto e sala).

Para isso, foi emulado um dispositivo móvel *Nexus* 4, da fabricante *Google*, com o auxílio do *software Android Studio* 1.3.2. Com o aplicativo aberto no dispositivo, foram cadastradas informações dos equipamentos, *Cores* e cômodos no banco de dados da aplicação, que segue o

modelo das Tabelas 3, 4 e 5, introduzidas mais à frente na seção “6.5.1. Acesso ao banco de dados”.

Um circuito utilizando um *Core* e um *Photon*, seguindo o modelo apresentado na Figura 3 da seção “4. O circuito de acionamento” foi implementado para atender as solicitações provenientes do aplicativo, confirmando o controle dos equipamentos conectados ao *kit*.

As imagens dessa seção foram extraídas do *display* do dispositivo emulado, validando o funcionamento do projeto da forma esperada.

Dito isso, as informações dessa seção têm o objetivo de apresentar o funcionamento completo do aplicativo desenvolvido.

O aplicativo apresenta duas janelas principais (também chamadas de telas ou *Activities* ao longo desse capítulo), que servem como *interface* para o usuário controlar os equipamentos domésticos.

A primeira janela (Figura 4) é representada pela classe *MainActivity* e apresenta os ambientes (cômodos) cadastrados da residência em ordem alfabética na forma de lista. O *menu* dessa *Activity* permite o gerenciamento de ambientes e *Cores*.

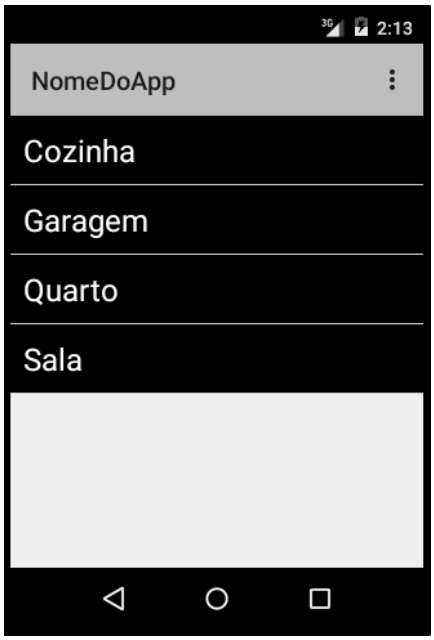


Figura 4 – *MainActivity*: Ambientes cadastrados

Ao clicar em um item da lista, o usuário é levado à segunda *Activity* da aplicação, a *ItemListActivity* (Figura 5). Essa *Activity* apresenta os equipamentos cadastrados referentes ao cômodo escolhido na tela anterior (*MainActivity*), também em ordem alfabética. Ao clicar em um item da lista, uma requisição para alterar o valor atual do equipamento é enviada ao *Core*. Quando recebida uma resposta positiva, a cor do texto e do fundo da célula selecionada são alteradas para representar o valor atual da saída do *Core* (demonstrado no item “Luz” da Figura 6). O *menu* dessa *Activity* permite o gerenciamento de equipamentos do cômodo em questão.

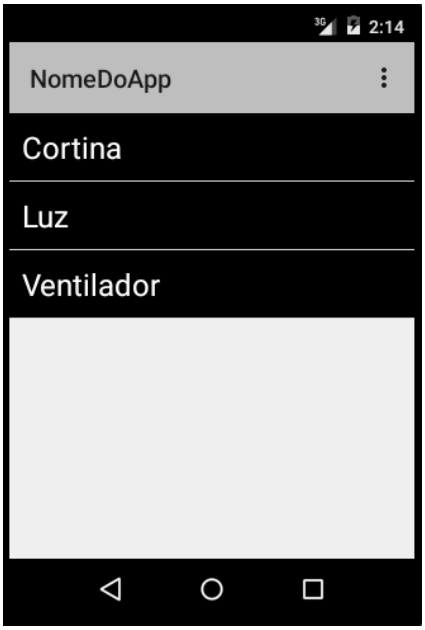


Figura 5 - *ItemListActivity*: Equipamentos cadastrados

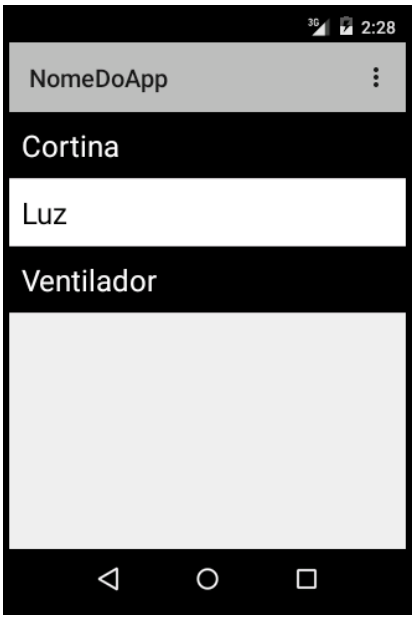


Figura 6 - *ItemListActivity* com o item “Luz” selecionado

Os *menus* de ambas as janelas estão disponíveis através do ícone de *menu* à extrema direita da barra de tarefas superior da aplicação ou através do botão físico de *menu* presentes em alguns dispositivos *Android*. As Figuras 7 e 8 apresentam, respectivamente, o *menu* da *MainActivity* e o *menu* da *ItemListActivity*.

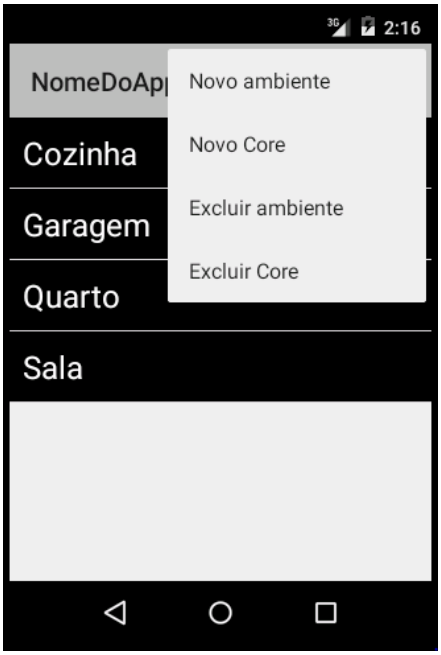


Figura 7 - *Menu* da *MainActivity*

Figura 8 - *Menu* da *ItemListActivity*

Ao selecionar um dos itens desses *menus*, uma caixa de diálogo é aberta para se comunicar com o usuário. Quando o usuário solicita a exclusão de ambientes, *Cores* ou equipamentos, a caixa de diálogo tem a aparência da Figura 9. Se o usuário, no entanto, solicitou a adição de algum desses itens, a caixa de diálogo tem a aparência da Figura 10.

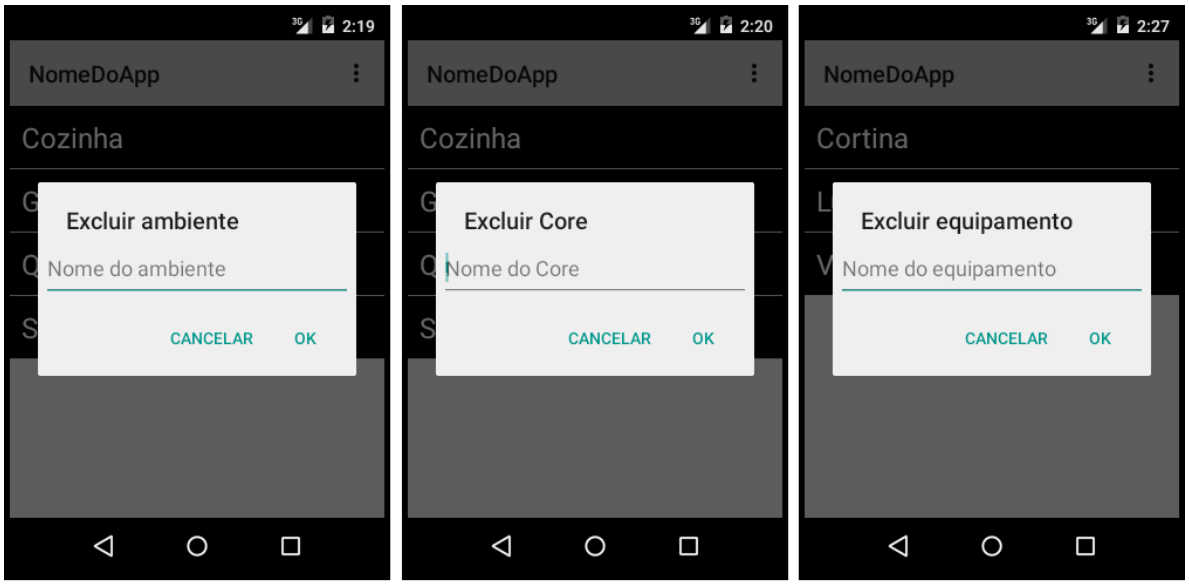
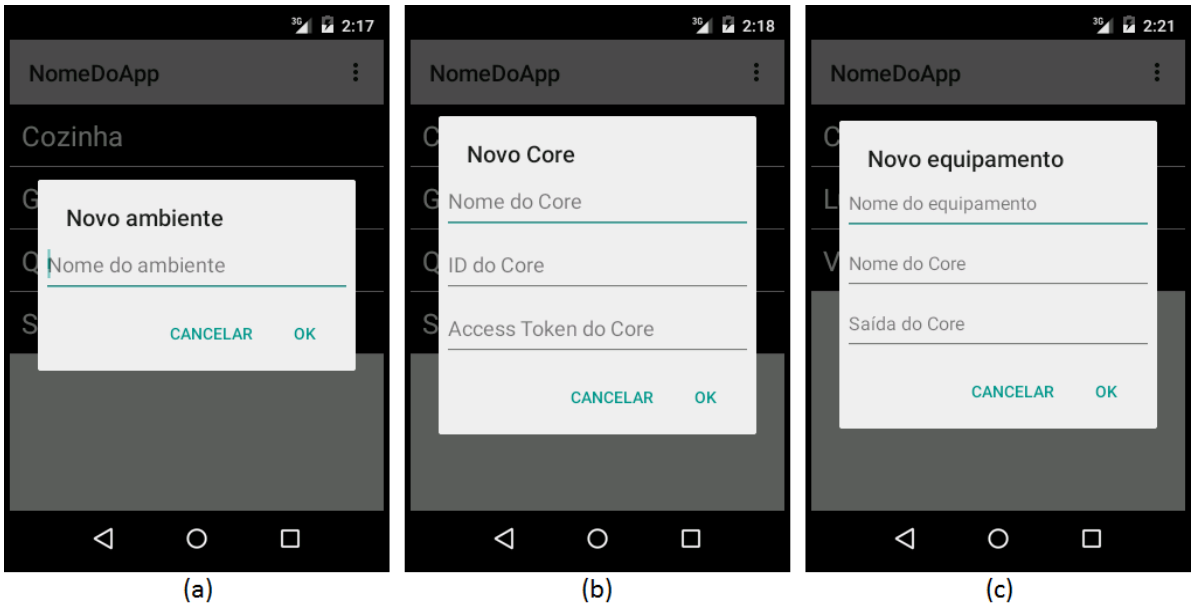


Figura 9 - Exclusão de (a) ambientes, (b) *Cores* ou (c) equipamentos



(a) (b) (c)

Figura 10 - Adição de (a) ambientes, (b) *Cores* ou (c) equipamentos

Na Figura 10.c, o campo para escolher o *Core* e a sua saída apresentam, assim que o usuário começa a digitar, uma lista, filtrada pelas letras iniciais, com os *Cores* cadastrados e as possíveis saídas que podem ser escolhidas, como demonstrado para o caso das saídas na Figura 11. O mesmo acontece com os campos para exclusão de itens representados na Figura 9.

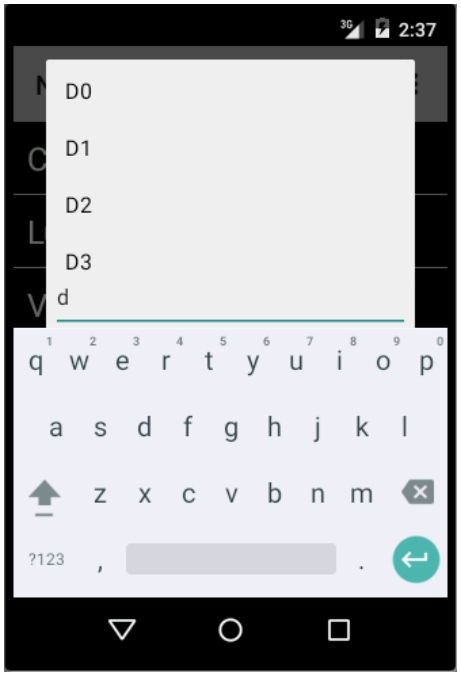


Figura 11 - Lista de possíveis saídas a selecionar

Em qualquer uma das caixas de diálogo, se o usuário tentar executar uma operação inválida como, por exemplo, tentar criar um ambiente com um nome já cadastrado ou cadastrar um equipamento a um *Core* inexistente, uma mensagem de texto é exibida na tela informando o erro, como ilustrado na Figura 12.

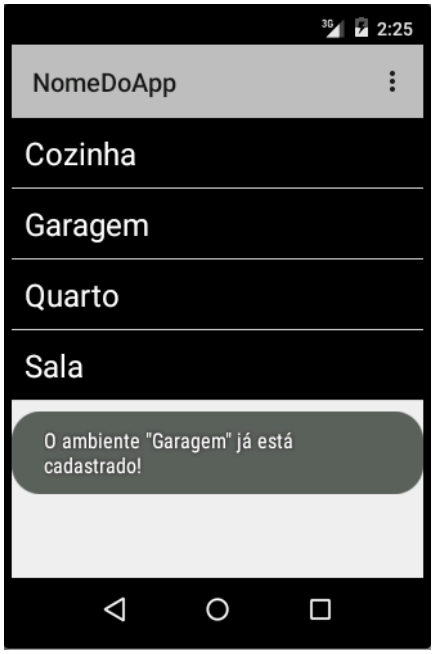


Figura 12 - Mensagem de erro

6.4. Ciclo de operação e classes da aplicação

Para que o aplicativo funcione da forma descrita na seção “6.3. Funcionamento do aplicativo”, foram desenvolvidas oito classes, que seguem o fluxo de operação representado na Figura 13.

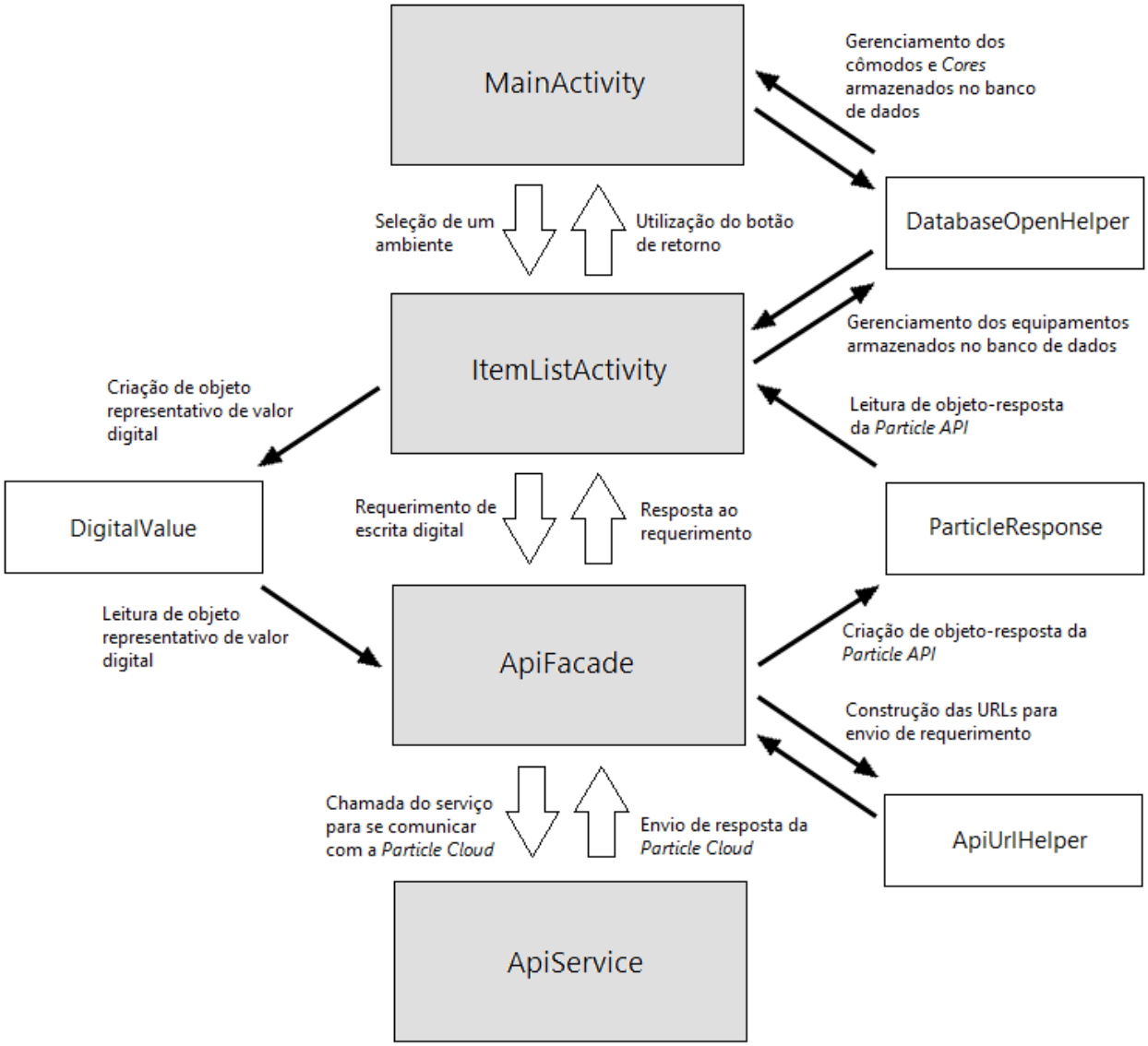


Figura 13 - Fluxo de operação do aplicativo

As quatro classes principais, em destaque na figura, foram utilizadas para realizar a *interface* com o usuário (*MainActivity* e *ItemListActivity*) e a comunicação com a *Particle Cloud* (*ApiFacade* e *ApiService*). Elas serão tratadas com mais detalhes nas seções “6.5. *Interface* com o usuário” e “6.6. Requerimentos para a *Particle Cloud*”. Além dessas, as outras quatro classes auxiliares apresentadas na figura foram criadas para facilitar a manipulação dos dados e ajudar na realização das operações necessárias para o funcionamento da aplicação. Suas funções estão relacionadas na Tabela 1.

Tabela 1 - Classes de Auxílio

Nome da classe	Função
ParticleResponse	Classe genérica que representa uma resposta da *Particle Cloud*
DatabaseOpenHelper	Subclasse de *SQLiteOpenHelper*, do pacote *android*.*database*.*sqlite*, utilizada para acessar o banco de dados *SQLite* do aplicativo
ApiUrlHelper	Classe genérica utilizada para auxiliar na construção de *URLs* para as requisições do tipo *REST*
DigitalValue	Classe genérica que representa o valor digital de uma saída do *Core*

Ao longo desse capítulo serão referenciadas algumas classes da *Android* API, descritas com mais detalhes na Tabela 2. A documentação detalhada da *Android* API pode ser encontrada na página oficial de desenvolvimento para *Android*, “http://developer.android.com/” (ANDROID REFERENCE, 2015).

Tabela 2 - Classes da *Android* API

Nome da classe	Função
Activity	Classe do pacote *android*.*app* que representa uma janela através da qual o usuário por interagir com o aplicativo
ListActivity	Subclasse de *Activity*, do pacote *android*.*app*, que apresenta no *display* uma lista de elementos
ResultReceiver	Classe do pacote *android.os* que representa uma *interface* genérica para receber uma resposta de algum objeto
LocalBroadcastManager	Classe do pacote *android*.*support*.*v4*.*content* utilizada para transmitir *Intents* dentro da aplicação a que pertence
Intent	Classe do pacote *android*.*content* que representa uma descrição abstrata de uma operação a ser realizada
BroadcastReceiver	Classe do pacote *android*.*content* utilizada para receber *Intents* que tenham sido transmitidas dentro ou fora da aplicação

Continua

Conclusão

Nome da classe	Função
Bundle	Classe do pacote *android.os* utilizada para armazenar um mapa de variáveis
IntentService	Classe do pacote *android.app* utilizada para lidar com requerimentos assíncronos (expressados como *Intents*) sob demanda
HttpURLConnection	Classe do pacote *java.net* utilizada para enviar e receber dados através da *internet*
Dialog	Classe do pacote *android.app* utilizada para criar uma caixa de diálogo para interação com o usuário em frente à atual *Activity* em exibição
SQLiteDatabase	Classe do pacote *android.database.sqlite* com métodos utilizados para gerenciar um banco de dados *SQLite*

6.5. *Interface* com o Usuário

Esta seção descreve as duas classes utilizadas para a *interface* com o usuário, a *MainActivity* e a *ItemListActivity*, bem como seu relacionamento com o banco de dados da aplicação.

6.5.1. Acesso ao banco de dados

Para que as informações referentes aos cômodos, *Cores* e equipamentos residenciais cadastrados sejam mantidas quando a aplicação for encerrada e possa ser resgatada quando a aplicação for reiniciada, é utilizada a integração do *Android* com o *software* para banco de dados embarcados *SQLite*.

O *SQLite* é um sistema de gerenciamento de bancos de dados relacionais que, diferente de outros sistemas, não é uma ferramenta cliente-servidor. Ao invés disso, ele é embutido junto com a aplicação, armazenando o banco de dados inteiro em um único arquivo no dispositivo em

que o aplicativo for instalado. O código-fonte para o *SQLite* está em domínio público e mais informações podem ser encontradas no endereço “http://www.sqlite.org/” (SQLITE, 2015).

Para acessar esse banco de dados, as classes *MainActivity* e *ItemListActivity* utilizam a classe *DatabaseOpenHelper* (conferir Tabela 1). O *DatabaseOpenHelper* representa um banco de dados com três tabelas. A primeira tabela, exemplificada na Tabela 3, é utilizada para gerenciar os equipamentos residenciais controlados pelo aplicativo. A segunda, exemplificada na Tabela 4, é utilizada para gerenciar os *Cores* conectados ao aplicativo. Já a terceira, exemplificada na Tabela 5, é utilizada para gerenciar os ambientes cadastrados pelo usuário. As tabelas utilizadas como exemplo foram completadas com os valores necessários para a construção das janelas apresentadas anteriormente nas Figuras 4 e 5 da seção “6.3. Funcionamento do aplicativo”.

Tabela 3 - Equipamentos cadastrados

_ID	**ROOM_NAME**	**ITEM_NAME**	**CORE_ID**	**CORE_PIN**
0	Sala	Cortina	999999999	D3
1	Garagem	Portão	000000000	D0
2	Sala	Luz	999999999	D6
3	Sala	Ventilador	999999999	D5

Tabela 4 - *Cores* cadastrados

CORE_ID	**CORE_NAME**	**CORE_ACCESS_TOKEN**
999999999	Core 1	abcdefgh
000000000	Core 2	hgfedcba

Tabela 5 - Ambientes cadastrados

_ID	ROOM_NAME
0	Cozinha
1	Garagem
2	Quarto
3	Sala

Através dos métodos *query()* e *insert()* da classe *SQLiteDatabase* (conferir Tabela 2) é possível, respectivamente, resgatar ou escrever informações no banco de dados da aplicação como os nomes dos cômodos, *Cores* e equipamentos residenciais conectados.

6.5.2. *MainActivity*

A *MainActivity* é uma subclasse de *ListActivity* (conferir Tabela 1), que dispõe no *display* os cômodos cadastrados no aplicativo em forma de lista. Durante o método *onCreate()*, chamado quando a janela é criada e está prestes a surgir no *display*, é feita a leitura dos cômodos cadastrados no banco de dados da aplicação. A lista de cômodos é, então, transmitida para a *Activity* através do método *setListAdapter()*.

No objeto visual que representa a lista em questão, é configurada uma ação para ser executada quando um dos elementos for selecionado pelo usuário através do método *setOnItemClickListener()*. O nome do cômodo da célula em questão é adicionado como um dado extra na *Intent* (conferir Tabela 2) que representa a operação de abertura da janela de equipamentos residenciais (a *ItemListActivity*). A *Intent* é passada como parâmetro através do método *startActivity()*.

O menu da *MainActivity* é configurado através dos métodos *onCreateOptionsMenu()* e *onOptionsItemSelected()*. O primeiro utiliza o arquivo de formato XML “menu_main.xml” salvo no diretório “/res/menu/” para configurar os itens pertencentes ao *menu* e o segundo adiciona ações a serem executadas quando o usuário seleciona uma das opções. Essas ações são responsáveis pela abertura de uma caixa de diálogo para o usuário interagir com o aplicativo, adicionando ou deletando novos *Cores* ou cômodos ao banco de dados da aplicação.

As caixas de diálogo são representadas pela classe *Dialog* (conferir Tabela 2) e utilizam os arquivos “dialog_add_room.xml”, “dialog_add_core.xml”, “dialog_delete_room.xml” e “dialog_delete_core.xml” salvos no diretório “/res/layout/” para configurar seus leiautes. Os dados que o usuário inserir nessas caixas de diálogo são atualizados no banco de dados da aplicação e na interface principal da *MainActivity* assim que a caixa de diálogo for encerrada.

6.5.3. *ItemListActivity*

A *ItemListActivity* é uma subclasse de *ListActivity* (conferir Tabela 2), que dispõe no *display* os equipamentos residenciais cadastrados no aplicativo referentes ao cômodo escolhido na *Activity* anterior (a *MainActivity*) na forma de lista. Durante o método *onCreate()*, chamado quando a janela é criada e está prestes a surgir no *display*, é feita a leitura dos equipamentos cadastrados no banco de dados da aplicação. A lista de equipamentos é, então, transmitida para a classe através do método *setListAdapter()*.

No objeto visual que representa a lista em questão, é configurada uma ação para ser executada quando um dos elementos for selecionado pelo usuário através do método *setOnItemClickListener()*. Essa ação deve buscar no banco de dados da aplicação o ID (código de identificação) e o pino do *Core* em que o equipamento selecionado está conectado, enviando uma requisição para a classe *ApiFacade* para alterar o valor da saída.

Para isso, antes dessa ação ser configurada, é criada uma nova instância (ou capturada a atual) da classe *ApiFacade*. É também configurado um *BroadcastReceiver* (conferir Tabela 2) para receber as mensagens que forem transmitidas pela classe *ApiFacade* endereçadas à classe *ItemListActivity*.

Através dessas mensagens, tratadas durante o método *onReceive()* do *BroadcastReceiver*, é possível recuperar um objeto da classe *ParticleResponse* (conferir Tabela 1) que irá identificar os detalhes da resposta recebida. Nesse método, a aplicação faz novamente uma consulta ao banco de dados para identificar qual equipamento corresponde aos dados da resposta, atualizando no *display* a célula referente ao equipamento com recursos visuais que identifiquem o novo valor da saída digital do *kit*.

Assim como na *MainActivity*, o menu dessa *Activity* é configurado através dos métodos *onCreateOptionsMenu()* e *onOptionsItemSelected()*. O primeiro utiliza o arquivo de formato XML

“menu_item_list.xml” salvo no diretório “/res/menu/” para configurar os itens pertencentes ao *menu* e o segundo adiciona ações a serem executadas quando o usuário seleciona uma das opções. Essas ações são responsáveis pela abertura de uma caixa de diálogo para o usuário interagir com o aplicativo, adicionando ou deletando novos equipamentos residenciais ao banco de dados da aplicação.

As caixas de diálogo são representadas pela classe *Dialog* (conferir Tabela 2) e utilizam os arquivos “dialog_add_item.xml” e “dialog_delete_item.xml” salvos no diretório “/res/layout/” para configurar seus leiautes. Os dados que o usuário inserir nessas caixas de diálogo são atualizados no banco de dados da aplicação e na interface principal da *ItemListActivity* assim que a caixa de diálogo for encerrada.

6.6. Requerimentos para a *Particle Cloud*

Esta seção descreve as duas classes utilizadas para a realização dos requerimentos para a *Particle Cloud*, a *ApiFacade* e a *ApiService*.

6.6.1. *ApiFacade*

A classe *ApiFacade* funciona como uma *interface* para enviar requerimentos para a *Particle Cloud* e transmitir as respostas recebidas através da aplicação.

A classe deverá ser inicializada durante o método *onCreate()* da *Activity ItemListActivity*, que é chamado sempre que a *Activity* é criada. Nessa *Activity*, quando o usuário selecionar um equipamento, o método *ApiFacade.digitalWrite()* deve ser invocado, recebendo como argumentos o ID (código de identificação) do *Core*, o pino que deverá ser acionado e o novo valor a ser registrado na sua saída.

Esse método cria um *ResultReceiver* (conferir Tabela 2) para receber a resposta do *Core* e transmite-a utilizando um *LocalBroadcastManager* (conferir Tabela 2). Para isso, ele utiliza o método *ApiService.post()*, passando como parâmetros um vetor de *Strings* (conjunto de caracteres) que representa a URL do *Core* e a operação desejada, um arquivo do tipo *Bundle*

(conferir Tabela 2) com as informações do pino a ser acionado e o novo valor desejado, e o *ResultReceiver* criado.

A resposta do *Core* é recebida através do método *onReceiveResult()* do *ResultReceiver* e armazenada em um objeto da classe *ParticleResponse* (conferir Tabela 1), para então ser transmitida para a classe *ItemListActivity* (que realiza a interface gráfica com o usuário) pelo *LocalBroadcastManager*, finalizando o ciclo de operação da classe *ApiFacade*.

6.6.2. *ApiService*

A classe *ApiService* é uma subclasse de *IntentService* (conferir Tabela 2). Ela é iniciada conforme necessário, atendendo cada *Intent* (conferir Tabela 2) em uma *thread* (linha ou encadeamento de execução) de trabalho separada da *thread* principal; e é pausada automaticamente quando fica sem trabalho. Esse padrão de processamento em fila é comumente usado para não sobrecarregar a *thread* principal (de interface com o usuário) com muitas tarefas.

A classe *ApiService* deve lidar com requerimentos assíncronos através do método *onHandleIntent()*. Nesse método, a *ApiService* utiliza como auxílio a classe *ApiUrlHelper* (conferir Tabela 1) para construir uma URL do tipo *post*, que chamará a função do *firmware* responsável por alterar a saída digital do *Core*.

Essa URL, juntamente com os dados provenientes da *ApiFacade* que identificam a ação a ser executada, são enviados para a *Particle Cloud* utilizando o protocolo HTTP. Para tanto, é utilizada a classe *HttpUrlConnection* (conferir Tabela 2). A resposta ao requerimento é então entregue ao *ResultReceiver* que havia sido passado como parâmetro pela classe *ApiFacade*, finalizando o ciclo de operação desse serviço.

6.7. *Logcat*

Caso o leitor tenha interesse em acompanhar o funcionamento do aplicativo, mensagens de registro são exibidas no *logcat*. O *logcat* é uma janela do *Android Studio* utilizada para visualizar e filtrar mensagens da aplicação enquanto ela está sendo executada através do emulador do programa.

Elas identificam, dentre outras coisas, a resposta da *Particle Cloud* para os requerimentos realizados durante a aplicação e sinalizam erros que podem ter ocorrido durante a comunicação com o servidor.

Para isso, o leitor deve executar o programa disponível no endereço “https://github.com/acfelipe/my-app” (CONTROLE [...], 2015) no *Android Studio* em um dispositivo emulado com *Android* 5.0 ou superior.

7. Conclusões

7.1. A Particle e futuras otimizações

A utilização dos produtos da *Particle* nesse trabalho permitiu aproveitar toda a estrutura do sistema oferecido pela empresa para realizar a conexão de objetos físicos com a *internet*. Além disso, os meios de comunicação que a empresa oferece para aprendizado (documentação, *templates* de *software*, ferramentas de desenvolvimento e fórum) auxiliaram na rápida absorção dos conceitos por trás da operação dos *kits* de desenvolvimento.

A *Particle*, como descrito anteriormente na seção “3. A *Particle*”, é uma empresa muito nova e seus produtos se encontram em constante atualização. Por ser um projeto de código aberto, há muito interesse de diversas empresas e parceiros em contribuir com a otimização dos sistemas envolvidos na comunicação com os dispositivos da *Particle*. Por isso, existe uma chance de que as versões das funções utilizadas nesse trabalho para comunicação com a *Particle Cloud* estejam depreciadas no momento em que o leitor estiver acompanhando esse trabalho. Essa, no entanto, não deve ser uma notícia ruim, pois essas alterações visam sempre eliminar eventuais problemas e tornar o sistema cada vez mais robusto.

Uma das grandes vantagens de projetos de código aberto é a disponibilização de bibliotecas, documentações e modelos de código que podem ser otimizados constantemente pela comunidade de desenvolvedores interna ou externa da empresa. Por isso, bibliotecas nativas como as do *Particle Android* SDK, introduzida na seção “6.2. O *Particle Android* SDK”, são sempre uma ótima opção para resolução dos problemas básicos envolvidos na comunicação com os produtos da *Particle*. Uma futura otimização do projeto pode incluir a utilização dessas bibliotecas, que atualmente se encontram em fase *beta* de desenvolvimento.

7.2. Possíveis melhorias para o trabalho futuro

O sistema desenvolvido nesse trabalho apresenta algumas fragilidades que devem ser corrigidas em versões posteriores.

Uma das fragilidades é não gravar o valor atual das saídas do *Core* quando o usuário navega pelas telas do aplicativo. Suponha que o usuário, por exemplo, tenha ligado a luz da

sala na tela de equipamentos residenciais (conferir Figuras 5 e 6) e depois retornado à tela de cômodos (conferir Figura 4). Quando o usuário selecionar novamente a sala na tela de cômodos, o item da lista correspondente à luz não mais indicará que ela se encontra ligada.

Existem algumas formas de corrigir essa fragilidade, como, por exemplo, salvar o valor atual das saídas do *Core* antes de fechar a tela de equipamentos residenciais. Essa solução, no entanto, não prevê que o equipamento pode ter seu estado alterado quando o usuário não estiver utilizando o aplicativo.

A melhor solução nesse caso é, portanto, reler o valor das saídas do *Core* correspondentes ao cômodo em questão toda vez que a tela referente àquele cômodo for aberta. Para isso, dois caminhos são possíveis: utilizar a função do *firmware* *myAppDigitalRead()* para realizar a leitura da saída (conferir o código da função no Apêndice C); ou expor uma variável do *firmware* para a *Particle Cloud* que represente a saída em questão.

No primeiro caso, a função deve ser invocada utilizando um requerimento do tipo *post* e deve retornar o valor da saída requisitada. No segundo caso, a variável deve ser do tipo “int”, “double” ou “string” e deve ser lida utilizando um requerimento do tipo *get*. Atualmente, um número máximo de 4 funções e 10 variáveis podem ser registradas na *Particle Cloud*. Esses números devem ser levados em conta caso o leitor tenha interesse em construir um sistema mais sofisticado.

Uma segunda fragilidade é a incapacidade do aplicativo de reconhecer alterações nas saídas do *Core* que não tenham sido requisitadas por ele. Se um usuário, por exemplo, acender uma lâmpada conectada à uma das saídas, outro usuário que esteja com o aplicativo aberto não irá notar essa alteração, pois ela não será comunicada instantaneamente para ele.

Novamente, existem algumas soluções possíveis, como constantemente requisitar o valor das saídas do *Core* quando estiver navegando no aplicativo. A melhor solução, entretanto, utiliza o sistema de eventos da *Particle*. No *firmware*, é possível publicar eventos para a *Particle Cloud* que serão encaminhados para todos os dispositivos que se registraram para acompanhar o evento em questão quando o mesmo for disparado. Esse processo envolve uma série de passos elucidados na subseção “*Cloud Functions*” da seção “*Firmware*”, disponível no endereço “https://docs.particle.io/reference/firmware/core/” (PARTICLE DEVICE FIRMWARE, 2015).

Nenhuma das fragilidades mencionadas, no entanto, afetam a comunicação do aplicativo com o *Core*. Em todos os casos, o usuário continua sendo capaz de controlar os

equipamentos que tiver cadastrado. É conveniente mencionar que, no caso da interrupção de energia para o *Core*, as saídas do *kit* permanecem salvas e voltam aos seus estados anteriores quando a energia for reestabelecida.

7.3. Pontos fortes do projeto

Esse trabalho não foi desenvolvido para ser estático e imutável, desempenhando apenas a funcionalidade de controlar equipamentos residenciais da forma como foi descrita nas seções anteriores. Ao invés disso, sempre se preservou a ideia de que futuras otimizações serão implementadas, inclusive aquelas que poderiam incluir funcionalidades completamente novas à aplicação, como a de realizar outros tipos de requerimento e até mesmo a total mudança da sua interface com o usuário.

Por isso, sua estrutura foi idealizada unindo conceitos de segmentação de funcionalidades compartilhados por desenvolvedores da *Particle* e *insights* criativos para a adaptação do *software*, permitindo a implementação de novos recursos sem que recursos atualmente presentes sejam impactados de forma negativa.

A classe *ApiFacade*, por exemplo, foi desenvolvida para funcionar como uma “fachada” para realizar requisições para a *Particle Cloud*. Todo e qualquer tipo de requisição deve utilizar uma única instancia dessa classe para enviar os comandos para a classe *ApiService*, que irá lidar com esses requerimentos em uma única *thread* (linha ou encadeamento de execução) de comandos separada da *thread* principal (de *interface* com o usuário). Se novos comandos forem necessários em versões futuras do aplicativo, basta adicionar métodos nessas classes que desempenhem as funções requisitadas. Da mesma forma, as classes *DigitalValue* e *ParticleResponse* podem ser adaptadas para agregar novos tipos de resposta da *Particle Cloud*, como requisições para as saídas analógicas do *kit*, não utilizadas nesse trabalho.

As classes para *interface* com o usuário, por sua vez, são completamente livres de comandos inerentes ao funcionamento do sistema da *Particle*. Elas podem, assim, ser modificadas sem que seja necessária a alteração de muitas linhas de código. Da mesma forma, o banco de dados pode ser modificado para agregar outros tipos de informação sem perder as informações básicas de cômodos, *Cores* e equipamentos cadastrados pelo usuário.

Finalmente, a classe *ApiUrlHelper* auxilia no processo de criação de uma *URL* para requisições para a *Particle Cloud* sem poluir o código da classe *ApiFacade*.

Dito isso, as atualizações citadas na seção “7.2. Possíveis melhorias para o trabalho futuro”, podem ser realizadas de forma muito mais simples e segura, sem a necessidade de alteração da estrutura principal do aplicativo.

Além dessa estrutura de classes, o texto dos títulos, botões e dicas das caixas de diálogo e dos itens dos menus da aplicação foram salvas no arquivo de formato XML “strings.xml” salvo no diretório “/res/values/”. Dessa forma, o aplicativo pode ser traduzido para outras línguas além do português bastando, para isso, adicionar o arquivo “strings.xml” com seus valores traduzidos dentro de um novo diretório como, por exemplo “/res/values-es/”.

7.4. Disciplinas de apoio

O curso “Engenharia Elétrica – Ênfase em Eletrônica” oferecido pela Escola de Engenharia de São Carlos (EESC) da Universidade de São Paulo (USP) foi fundamental para a obtenção dos conhecimentos básicos necessários para todo o desenvolvimento desse trabalho.

Conceitos básicos de programação foram inseridos no primeiro e segundo semestres do curso, através das disciplinas “Introdução à Ciência da Computação” e “Linguagens de Programação e Aplicação”.

As disciplinas “Sistemas Digitais”, “Laboratório de Sistemas Digitais I e II”, “Tópicos Especiais em Sistemas Digitais” e “Aplicação de Microprocessadores I e II” forneceram um forte subsídio nos conceitos fundamentais por trás da lógica digital e das técnicas de projeto de sistemas microprocessados.

A disciplina optativa “Projeto de Sistemas Digitais”, ministrada durante o sétimo período do curso pelo professor Evandro Luís Linhari Rodrigues permitiu uma primeira aproximação com o desenvolvimento para dispositivos *Android*. Durante a disciplina, em conjunto com a aluna Andrea Pineda, foi desenvolvido um mural eletrônico, capaz de buscar notícias de diversos portais diferentes e atualizá-las na tela do *kit* de desenvolvimento utilizado, com a possibilidade de conexão do *kit* com um televisor para melhor leitura das informações.

Posteriormente, durante intercâmbio realizado na instituição de ensino *Dublin City University* (DCU), na cidade de Dublin, Irlanda, a disciplina "*Object-Oriented Programming with Embedded Systems*" fortaleceu o conhecimento do autor na disciplina de linguagens de programação orientada a objetos, fornecendo uma forte base para o desenvolvimento de programas em *Java*. Durante a disciplina, foi desenvolvido um sistema de *e-mails* em Java em que as mensagens deveriam ficar hospedadas em um servidor embarcado no *kit* de desenvolvimento *BeagleBone Black*.

Sem toda a base técnica provida por essas e outras disciplinas do curso, seria impossível a criação do sistema descrito nesse trabalho no que diz respeito tanto ao desenvolvimento do circuito de acionamento dos equipamentos residenciais e entendimento do funcionamento do *kit* de desenvolvimento utilizado quanto ao desenvolvimento do *firmware* e do aplicativo utilizados para controlar o *kit*.

REFERÊNCIAS[1]

ANDROID REFERENCE. Apresenta a documentação dos pacotes e classes utilizados no desenvolvimento para dispositivos Android. Disponível em: <http://developer.android.com/reference>. Acesso em: 07 dez. 2015.

ANDROID STUDIO. Disponibiliza o software Android Studio para download. Disponível em: <http://developer.android.com/sdk/>. Acesso em: 07 dez. 2015.

BERI, J. Spark Core pinout diagram. [2015?]. 1 gravura, color. Altura: 1358 pixels. Largura: 2484 pixels. 394Kb. Formato PNG. Disponível em: <https://docs.particle.io/assets/images/spark-pinout.png>. Acesso em: 02 nov. 2015.

BORTOLUZZI, M. Histórico da automação residencial. Revista SRA Engenharia, Santa Maria, 10 jan. 2013. Disponível em: <http://sraengenharia.blogspot.com.br/2013/01/historico-da-automacao-residencial_10.html>. Acesso em: 02 nov. 2015.

CONTROLE ON/OFF de equipamentos residenciais com o kit de desenvolvimento Particle Core. Projeto desenvolvido por F. Carvalho. Disponível em: <https://github.com/acfelipe/my-app>. Acesso em: 07 dez. 2015.

FREERTOS. Disponível em: <http://www.freertos.org/>. Acesso em: 07 dez. 2015.

GARTNER. Gartner says the Internet of Things installed base will grow to 26 billion units by 2020. Stamford, CT, 12 dez. 2013. Disponível em: <http://www.gartner.com/newsroom/id/2636073>. Acesso em: 02 nov. 2015.

GETTING STARTED. Apresenta os primeiros passos para a conexão do Particle Core com a internet. Disponível em: <https://docs.particle.io/guide/getting-started/start/core/>. Acesso em: 07 dez. 2015.

[1] De acordo com a Associação Brasileira de Normas Técnicas. NBR 6023.

MORGAN, J. A simple explanation of ‘the Internet of Things’. Forbes, Jersey City, 10 mai. 2014. Disponível em: <http://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/>. Acesso em: 02 nov. 2015.

MURATORI, J. R. Os desafios do mercado da Automação Residencial. AECweb, São Paulo, 2013. Disponível em: < http://www.aecweb.com.br/cont/a/os-desafios-do-mercado-da-automacao-residencial_8192>. Acesso em: 02 nov. 2015.

PARTICLE. Disponível em: <https://www.particle.io/>. Acesso em: 07 dez. 2015.

PARTICLE ANDROID REFERENCE. Apresenta a documentação do Particle Android SDK. Disponível em: <https://docs.particle.io/reference/android/>. Acesso em: 07 dez. 2015.

PARTICLE BUILD. Plataforma de desenvolvimento de software para dispositivos Particle. Disponível em: <https://build.particle.io>. Acesso em: 07 dez. 2015.

PARTICLE DEVICE FIRMWARE. Apresenta a documentação para o desenvolvimento do firmware para o dispositivo Particle Core. Disponível em: <https://docs.particle.io/reference/firmware/core/>. Acesso em: 07 dez. 2015.

PARTICLE SOURCE. Apresenta designs de hardware, firmware, cloud software, templates de aplicativos para dispositivos móveis e ferramentas de desenvolvimento necessários para a criação de dispositivos conectados com a internet. Disponível em: <http://spark.github.io/>. Acesso em: 07 dez. 2015.

ROTHFELD, L. The smart home. In: CONSUMER ELECTRONICS SHOW, 2015, Las Vegas. Tech Time Machine. Disponível em: <http://mashable.com/2015/01/08/smart-home-tech-ces/>. Acesso em: 02 nov. 2015.

SPARK CORE FOR ANDROID. Disponibiliza o download do aplicativo Spark Core para Android. Disponível em: <https://play.google.com/store/apps/details?id=io.spark.core.android>. Acesso em: 07 dez. 2015.

SPARK CORE FOR IPHONE. Disponibiliza o download do aplicativo Spark Core para iPhone. Disponível em: <https://itunes.apple.com/us/app/spark-core/id760157884>. Acesso em: 07 dez. 2015.

SQLITE. Disponível em: <http://www.sqlite.org/>. Acesso em: 07 dez. 2015.

TECHINBRAZIL. Integradores de automação residencial no Brasil. São Paulo, SP, 22 jun. 2015. Disponível em: <https://techinbrazil.com.br/integradores-de-automacao-residencial-no-brasil>. Acesso em: 02 nov. 2015.

WIRING REFERENCE. Apresenta a documentação do framework Wiring. Disponível em: <http://wiring.org.co/reference/>. Acesso em: 07 dez. 2015.

WORTMEYER, C.; FREITAS, F.; CARDOSO, L. Automação residencial: busca de tecnologias visando o conforto, a economia, a praticidade e a segurança do usuário. II Simpósio de Excelência em Gestão e Tecnologia, Resende, 2005. Disponível em: <http://www.aedb.br/seget/arquivos/artigos05/256_SEGET%20-%20Automacao%20Residencial.pdf>. Acesso em: 02 nov. 2015.

APÊNDICE A - Informações técnicas do *Particle Core*

Na seção “2.1. O *Particle Core*” foram fornecidas informações gerais a respeito do *kit* de desenvolvimento utilizado nesse trabalho. Este apêndice é dedicado a descrever com mais detalhes as características principais relacionadas com o *hardware* do produto.

O *Particle Core* possui 18 pinos para troca de dados com componentes externos, sendo 8 analógicos (A0 até A7), 8 digitais (D0 até D7) e 2 pré-configurados para comunicação serial (TX e RX). Cada um deles opera com tensão de 3,3V, com exceção dos pinos D0, D1, D3, D4, D5, D6 e D7, que são tolerantes a uma tensão máxima de 5V. O diagrama completo da pinagem do *kit* é ilustrado na Figura 14.

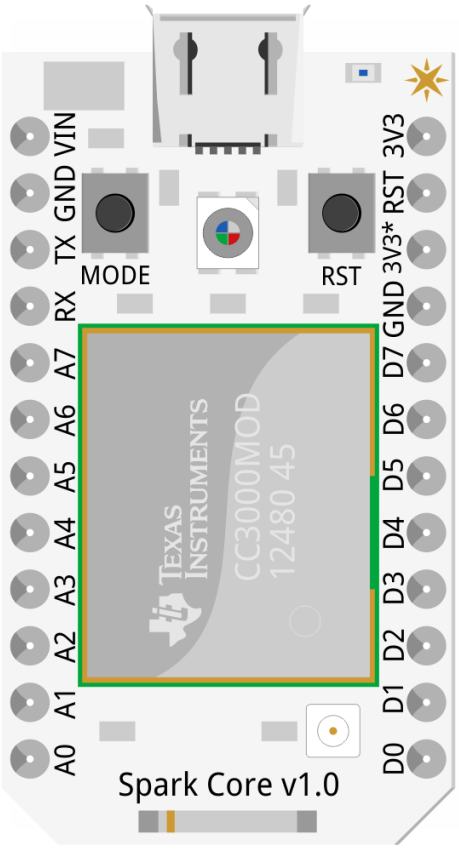


Figura 14 - Diagrama de pinagem do *Particle Core* (BERI, 2015)

Os pinos digitais, utilizados para comandar os relés utilizados nesse trabalho, podem ser configurados como entrada (com ou sem resistor *pull-up* ou *pull-down* – iguais a 40kΩ) ou como saída (*push-pull* ou dreno aberto). Cada um deles pode fornecer ou drenar uma corrente mínima de 8mA e máxima de 20mA.

Os pinos analógicos produzem um sinal PWM (*Pulse Width Modulation*) em que o ciclo de trabalho pode variar, fornecendo uma potência média total variável. O sinal tem uma resolução de 8 bits e opera na frequência de 500Hz. Como entrada, entretanto, o usuário pode ler valores com resolução de 12 bits.

O *Core* possui três portas seriais, uma CDC *(Communications Device Class)* disponível através da porta USB e duas USART *(Universal Synchronous Asynchronous Receiver Transmitter)*, uma disponível através dos pinos TX e RX e outra disponível via D1 (Tx) e D0 (Rx).

A comunicação serial do microcontrolador com outros componentes pode ser feita através do protocolo SPI (*Serial Peripheral Interface*), através dos pinos A2 (*Slave Select*), A3 (*Serial Clock*), A4 (*Master In Slave Out*) e A5 (*Master Out Slave In*) ou do protocolo I^2C (*Inter-Integrated Circuit*), através dos pinos D0 (*Serial Data Line*) e D1 (*Serial Clock*).

O pino V_{IN} pode ser utilizado para alimentação do *Core* (aceitando uma tensão entre 3,6 e 6V) ou, quando o *Core* é conectado via USB, para alimentação de componentes externos, fornecendo uma tensão de 5V. Além dele, o pino 3V3 gera uma saída de 3,3V não regulada, enquanto o pino 3V3* fornece o mesmo valor de tensão regulada.

Dos componentes de maior relevância que constituem o *Core*, podem-se citar o microcontrolador *STM32F103CB* baseado na *CPU ARM 32-bit Cortex™-M3*, o módulo *Wi-Fi TI CC3000*, o *chip* de memória *flash* externa *SST25VF016B* de 2MB, o regulador de tensão *Microchip MCP1825S-3302E* que converte a entrada de 3,6V a 6V para 3,3V e o circuito RF para captação do sinal de *Wi-Fi*.

Mais informações a respeito do *Particle Core* incluindo a folha de dados do produto podem ser encontradas no *website* oficial da *Particle*, “http://particle.io” (PARTICLE, 2015).

APÊNDICE B - Primeiros passos para a conexão do *Particle Core*

Este apêndice tem por objetivo guiar o leitor através do processo de conexão do *Particle Core* com a *internet* pela primeira vez. É recomendado que a primeira configuração do *kit* seja feita através do aplicativo para aparelhos móveis da *Particle*, disponível para aparelhos *iPhone* ou *Android*. O *download* pode ser realizado nos endereços abaixo:

- *iPhone*: “https://itunes.apple.com/us/app/spark-core/id760157884” (SPARK CORE FOR IPHONE, 2015);
- *Android*: “https://play.google.com/store/apps/details?id=io.spark.core.android” (SPARK CORE FOR ANDROID, 2015).

A forma mais fácil de alimentar o dispositivo é através da conexão do cabo USB em uma fonte que forneça 5V de saída, como a saída USB do computador. Assim que conectado à energia, o LED RGB posicionado na parte superior do *kit* deve começar a piscar na cor azul. Isso significa que o dispositivo está no modo de escuta, esperando as credenciais da rede *Wi-Fi* para se conectar à *internet*. Para forçar o dispositivo a entrar no modo de escuta, basta segurar o botão físico “MODE”, localizado na parte superior do *kit*, por três segundos.

Para realizar a reivindicação do *Core* e transmitir as credenciais da rede *Wi-Fi*, é necessário que o usuário tenha uma conta na *Particle*, que pode ser criada através do aplicativo mencionado anteriormente. Após o *login* no aplicativo, deve-se seguir as instruções na tela para finalizar a configuração.

Mais informações a respeito desse processo podem ser encontradas no endereço “https://docs.particle.io/guide/getting-started/start/core/” (GETTING STARTED, 2015).

APÊNDICE C - Código do *firmware*

Abaixo se encontra o código completo em *Wiring* do *firmware* utilizado nesse projeto.

```
>   // Declaração das funções do firmware
>
>   int myAppDigitalRead(String pin);
>   int myAppDigitalWrite(String command);
>   int myAppAnalogRead(String pin);
>   int myAppAnalogWrite(String command);
>
>
>   // Função chamada uma única vez quando o dispositivo inicializa
>
>   void setup()
>   {
>     // Exposição das funções do firmware para a Particle Cloud
>
>     Spark.function("digitalread", myAppDigitalRead);
>     Spark.function("digitalwrite", myAppDigitalWrite);
>     Spark.function("analogread", myAppAnalogRead);
>     Spark.function("analogwrite", myAppAnalogWrite);
>   }
>
>
>   // Função que roda continuamente durante a execução da aplicação
>   // Não é necessário colocar nada aqui
>
>   void loop(){}
>
>
>   // Lê o valor digital de um determinado pino
>   // Retorna o valor do pino (0 ou 1) ou um valor negativo em caso de erro
>
>   int myAppDigitalRead(String pin)
>   {
>     int pinNumber = pin.charAt(1) - '0';
>     if (pinNumber< 0 || pinNumber >7) return -1;          // Erro: pino inexistente.
>     if(pin.startsWith("D"))
>     {                                                     // Entra aqui se for pino digital
>       pinMode(pinNumber, INPUT_PULLDOWN);                 // Altera modo do pino para leitura
>       return digitalRead(pinNumber);                      // Retorna valor digital do pino
>     }
>     else if (pin.startsWith("A"))
>     {                                                     // Entra aqui se for pino analógico
>       pinMode(pinNumber+10, INPUT_PULLDOWN);              // Altera modo do pino para leitura
>       return digitalRead(pinNumber+10);                   // Retorna valor digital do pino
>     }
>     return -2;                                            // Erro de sintaxe
>   }
```

```
>
>
>     // Altera o valor digital de um determinado pino
>     // Retorna 1 em caso de sucesso ou um valor negativo em caso de erro
>
>     int myAppDigitalWrite(String command)
>     {
>       bool value = 0;
>       int pinNumber = command.charAt(1) - '0';
>       if (pinNumber< 0 || pinNumber >7) return -1;              // Erro: pino inexistente.
>       if(command.substring(3,7) == "HIGH") value = 1;
>       else if(command.substring(3,6) == "LOW") value = 0;
>       else return -2;                                           // Erro de sintaxe
>       if(command.startsWith("D"))
>       {                                                         // Entra aqui se for pino digital
>         pinMode(pinNumber, OUTPUT);                             // Altera modo do pino para escrita
>         digitalWrite(pinNumber, value);                         // Altera o valor digital do pino
>         return 1;                                               // Retorna 1: Sucesso
>       }
>       else if(command.startsWith("A"))
>       {                                                         // Entra aqui se for pino analógico
>         pinMode(pinNumber+10, OUTPUT);                          // Altera modo do pino para escrita
>         digitalWrite(pinNumber+10, value);                      // Altera o valor digital do pino
>         return 1;                                               // Retorna 1: Sucesso
>       }
>       else return -3;                                           // Erro de sintaxe
>     }
>
>
>     // Lê o valor analógico de um determinado pino
>     // Retorna o valor do pino (0 a 4095) ou um valor negativo em caso de erro
>
>     int myAppAnalogRead(String pin)
>     {
>       int pinNumber = pin.charAt(1) - '0';
>       if (pinNumber< 0 || pinNumber >7) return -1;              // Erro: pino inexistente.
>       if(pin.startsWith("D"))
>       {                                                         // Entra aqui se for pino digital
>         pinMode(pinNumber, INPUT);                              // Altera modo do pino para leitura
>         return analogRead(pinNumber);                           // Retorna valor analógico do pino
>       }
>       else if (pin.startsWith("A"))
>       {                                                         // Entra aqui se for pino analógico
>         pinMode(pinNumber+10, INPUT);                           // Altera modo do pino para leitura
>         return analogRead(pinNumber+10);                        // Retorna valor analógico do pino
>       }
>       return -2;                                                // Erro de sintaxe
>     }
```

```
>
>
>     // Altera o valor analógico de um determinado pino
>     // Retorna 1 em caso de sucesso ou um valor negativo em caso de erro
>
>     int myAppAnalogWrite(String command)
>     {
>       int pinNumber = command.charAt(1) - '0';
>       if (pinNumber< 0 || pinNumber >7) return -1;          // Erro: pino inexistente.
>       String value = command.substring(3);
>       if(command.startsWith("D"))
>       {                                                     // Entra aqui se for pino digital
>         pinMode(pinNumber, OUTPUT);                         // Altera modo do pino para escrita
>         analogWrite(pinNumber, value.toInt());              // Altera o valor analógico do pino
>         return 1;                                           // Retorna 1: Sucesso
>       }
>       else if(command.startsWith("A"))
>       {                                                     // Entra aqui se for pino analógico
>         pinMode(pinNumber+10, OUTPUT);                      // Altera modo do pino para escrita
>         analogWrite(pinNumber+10, value.toInt());           // Altera o valor analógico do pino
>         return 1;                                           // Retorna 1: Sucesso
>       }
>       else return -2;                                       // Erro de sintaxe
>     }
```