UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE ELETRÔNICA

ENGENHARIA ELETRÔNICA

CHRISTIAN BECKER PEPINO

GUILHERME DIAS

POCKET GUARD – DISPOSITIVO PORTÁTIL DE SEGURANÇA IOT

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2018

CHRISTIAN BECKER PEPINO

GUILHERME DIAS

POCKET GUARD – DISPOSITIVO PORTÁTIL DE SEGURANÇA IOT

Trabalho de Conclusão de Curso apresentado à disciplina de Trabalho de Conclusão de Curso 2, como requisito parcial para obtenção de grau de Engenheiro do curso de Engenharia Industrial Elétrica com ênfase em Eletrônica e Telecomunicações, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Daniel Fernando Pigatto

CURITIBA

2018

CHRISTIAN BECKER PEPINO
GUILHERME DIAS

POCKET GUARD – DISPOSITIVO PORTÁTIL DE SEGURANÇA IOT

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

Curitiba, 23 de fevereiro de 2018.

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

_____
Profª. Drª. Carmen Caroline Rasera
Responsável pelos Trabalhos de Conclusão de Curso
de Engenharia Eletrônica do DAELN

BANCA EXAMINADORA

Prof. Dr. Daniel Fernando Pigatto Universidade Tecnológica Federal do Paraná Orientador	Prof. Me. Daniel Rossato de Oliveira Universidade Tecnológica Federal do Paraná
Profª. Drª. Tânia Lúcia Monteiro Universidade Tecnológica Federal do Paraná	Profª. Drª. Keiko Verônica Ono Fonseca Universidade Tecnológica Federal do Paraná

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

RESUMO

PEPINO, Christian Becker. **DIAS**, Guilherme. **POCKET GUARD – DISPOSITIVO PORTÁTIL DE SEGURANÇA IOT**. 2018. 111 f. Trabalho de Conclusão de Curso - Engenharia Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2018.

Este projeto consiste no desenvolvimento de um sistema de rastreamento e monitoramento de pertences, de baixo custo, utilizando conceitos da Internet das Coisas. O sistema contém três elementos principais: um protótipo de rastreador, baseado na plataforma Arduino, conectado aos módulos acelerômetro, Bluetooth e GNSS/GSM; um aplicativo desenvolvido para iOS, com uma interface gráfica que permite ao usuário configurar e monitorar o protótipo; e, por último, um servidor online para fazer a interface entre o aplicativo e o protótipo. Os resultados obtidos neste trabalho comprovam a possibilidade de se obter um dispositivo preciso, robusto e de baixo custo, integrado à Internet das Coisas, que pode ser aplicado para prover segurança física para pertences pessoais.

Palavras-chave: Internet das Coisas. Segurança. Rastreamento. Monitoramento.

ABSTRACT

PEPINO, Christian Becker. **DIAS**, Guilherme. **POCKET GUARD – DISPOSITIVO PORTÁTIL DE SEGURANÇA IOT**. 2018. 111 f. Trabalho de Conclusão de Curso - Engenharia Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2018.

This project consists on the development of a low-cost system for tracking and monitoring personal belongings, using the concept of the Internet of Things. The system is composed of three separate parts: a prototype based on the Arduino platform, connected to Bluetooth, GNSS/GSM and accelerometer modules; an application developed for iOS with a graphical user interface that allows the prototype configuration and monitoring; and, lastly, a server to be the interface between the application and the prototype. Results prove the possibility of implementing an accurate, robust, and low-cost IoT-ready device, which can be used as a tracker for personal belongings.

Keywords: Internet of Things. Security. Tracking. Monitoring.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
BJSON	Binary Javascript Object Notation
BLE	Bluetooth Low Energy
DB	Database
DPSK	Differential Phase Shift Keying
DQPSK	Differential Quadrature Phase Shift Keying
GCD	Grand Central Dispatch
GPIO	General-purpose input/output
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global Systems for Mobile communications
HSPA	High Speed Packet Access
HTTP	Hypertext Transfer Protocol
I^2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IoT	Internet of Things
ISM	Industrial, Scientific and Medical
JSON	Javascript Object Notation
LLDB	Low Level Debugger
LTE	Long Term Evolution
OS	Operational System
RTOS	Real-Time Operating System
RX	Receiver
SCL	Serial Clock
SDA	Serial Data
SIG	Special Interest Group
SoC	System on a Chip
SQL	Structured Query Language

TX	Transmitter
URL	Uniform Resource Locator
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Segundo reportagem da Folha de São Paulo (2017), ocorre um roubo de carro no Brasil por minuto, de modo que, em 2016, foram roubados um total de 557 mil veículos no país. Este é apenas um exemplo da falta de segurança, que se estende a todo tipo de pertence pessoal no Brasil. Diante desta realidade, o presente trabalho procura desenvolver um protótipo de um dispositivo que aumente o controle e auxilie no monitoramento, seja de veículos ou de qualquer outro objeto pessoal, utilizando conceitos de Internet das Coisas para realizar a integração de bens pessoais a um ambiente inteligente e automatizado.

Hoje, apesar da evolução dos sistemas eletrônicos de segurança e monitoramento, da microeletrônica e das telecomunicações, existem poucas soluções práticas e portáteis para efetuar o monitoramento de pertences. Já existem, de fato, sistemas de segurança para imóveis, automóveis e objetos de luxo. São soluções com fórmulas conhecidas, que usam recursos como câmeras de segurança e rastreadores de posicionamento, e geralmente são fixas e fisicamente acopladas a seus objetos de interesse através de um processo de instalação manual, o que as torna, inevitavelmente, custosas.

O intuito deste trabalho é, então, dispensar este acoplamento fixo do sistema de monitoramento ao objeto de interesse, possibilitando “micromonitoramentos” a qualquer momento e em qualquer lugar. Por exemplo, o usuário poderá verificar se seu paletó/bolsa está sendo furtado de sua cadeira durante uma festa, se o manobrista está conduzindo seu automóvel pelas ruas sem a devida permissão, se as gavetas de seu escritório estão sendo reviradas enquanto encontra-se em reunião, se seu animal de estimação fugiu, se sua bicicleta continua estacionada onde a deixou, entre outras inúmeras possibilidades.

O sistema de monitoramento aqui desenvolvido propõe integrar-se à vida digital do usuário, participando da sua rotina em conjunto com outros dispositivos (como *smartphones*, *wearables* e *tablets*), trazendo mais tranquilidade no cuidado de seus

pertences. Pode vir a ser utilizado futuramente, também, por pessoas ou instituições que necessitem de uma solução barata para monitorar ambientes críticos de suas instalações internas, como por exemplo um data center, e ser monitorado pela própria rede, de modo a, por exemplo, acionar alarmes ou travas automaticamente em caso de atividades anormais.

A ideia nasce em uma época favorável à integração das “coisas” à Internet (a *Internet of Things*, ou Internet das Coisas), inovando na forma de monitorar itens de menor custo e que acabam sendo deixados de lado por fabricantes de dispositivos para este segmento do mercado. A próxima seção vai mostrar alguns exemplos de trabalhos que têm sido desenvolvidos com objetivos parecidos aos deste e as diferenças entre eles.

1.2 TRABALHOS RELACIONADOS

Os rastreadores Autotrac Mini (AUTOTRAC, 2018) e Apeggo (APEGGO GPS, 2018) operam baseados em localização. Para tal, contam com módulos GPS (*Global Positioning System*), aplicativos de celular para efetuar o monitoramento e planos de assinatura mensal ou anual. Além disso, exigem a compra do produto que é acoplado aos objetos/pessoas monitorados, o que representa um custo extra.

Outro trabalho relacionado é o rastreador TrackR (TRACKR, 2018), um chaveiro que, utilizando apenas Bluetooth, se vincula ao *smartphone* do usuário através de um aplicativo. Pela intensidade do sinal de comunicação, o *smartphone* consegue mensurar a distância aproximada do dispositivo. No entanto, se o dispositivo sair do alcance do *smartphone*, este envia uma mensagem ao servidor da TrackR indicando a identificação do dispositivo perdido, e este servidor repassa esta identificação para outros celulares que possuem o TrackR instalado na região. Quando um destes *smartphones* detecta o dispositivo perdido, envia uma mensagem contendo as coordenadas geográficas para o servidor, que as envia ao usuário original. Esta ideia de utilizar o GNSS (*Global Navigation Satellite System*, termo que engloba todas as tecnologias de posicionamento via satélite) dos *smartphones* em rede para encontrar

dispositivos que não possuem GNSS é chamado de “*Crowd* GPS”. No entanto, uma desvantagem natural é que em regiões com poucos usuários dificilmente o dispositivo perdido será encontrado.

Existem várias formas de se detectar potenciais atividades de furto (de objetos) ou de acidente (no caso de pessoas). Nos trabalhos de CHAN et al. (2009) e ZHENG et al. (2013), os autores propõem soluções que monitoram objetos registrados e equipados com RFID, localizados dentro da área de cobertura de uma central de monitoramento. O uso de tecnologias como o RFID ou o Bluetooth para identificar se um objeto está sendo afastado do dono podem apresentar uma relevante eficiência para boa parte das situações, mas não cobrem todos os casos. Por exemplo, uma pessoa não autorizada pode pegar um telefone celular, ter acesso a informações pessoais do dono e fazer tudo isso mantendo o dispositivo dentro da área de cobertura da central de monitoramento, assim, não gerando alerta nenhum. Um segundo exemplo pode ocorrer quando um idoso que usa um dispositivo de monitoramento sofre uma queda e não recebe atendimento devido à não identificação da ocorrência por um sistema baseado em comunicação de curto alcance.

Sendo assim, o uso de tecnologias de rede sem fio de curto alcance associadas a um acelerômetro representa um meio adequado de identificar o uso não autorizado de dispositivos e ocorrências com pessoas, além de identificar também possíveis furtos. Uma implementação com sensores de baixo custo e servidor próprio também auxiliam na redução do custo do sistema final, permitindo que a acessibilidade ao produto seja maior.

1.3 OBJETIVOS

O principal objetivo consiste em criar uma nova ferramenta de baixo custo para segurança de objetos ou sistemas pessoais, por meio de um dispositivo de monitoramento. Este aparelho contará com sensor de movimento (acelerômetro), módulo de comunicação GSM (*Global Systems for Mobile communications*), módulo de comunicação Bluetooth e módulo de GNSS. O dispositivo se conectará a um servidor

web que fará a interface entre ele e o *smartphone*, onde o usuário poderá configurar todos os seus dispositivos ou ser notificado quanto a eventos em sua rede de monitoramento.

O aplicativo desenvolvido visa a simplicidade e a praticidade, não sendo necessária nenhuma configuração no hardware em si, de modo que o usuário já possa utilizar o dispositivo de maneira intuitiva assim que o adquirir, considerando que o dispositivo venha a ser lançado no mercado.

Além disso, um objetivo complementar ainda consiste na preocupação com a redução do consumo de energia do protótipo de rastreamento, permitindo que ele seja usado por longos períodos de tempo acoplado a objetos de interesse.

1.4 ESTRUTURA DO TEXTO

Este trabalho está dividido em quatro capítulos. No primeiro capítulo, “Revisão da Literatura”, são estabelecidos conceitos, topologias e protocolos que foram utilizados para o desenvolvimento do projeto. No segundo capítulo, “Pocket Guard”, é apresentado um panorama de como o sistema funciona de modo geral, fundamentado nos conceitos vistos na seção anterior, e analisando todo processo de desenvolvimento de cada parte do projeto especificamente. No terceiro capítulo, “Resultados”, são realizados testes de cenários reais com o sistema e uma discussão sobre a usabilidade e o consumo de energia do protótipo implementado. Por fim, no capítulo “Conclusões”, efetua-se uma análise final da contribuição oferecida com o desenvolvimento deste trabalho, uma listagem das principais dificuldades encontradas e algumas sugestões de trabalhos futuros.

2 REVISÃO DA LITERATURA

2.1 CONSIDERAÇÕES INICIAIS

Este capítulo define o conceito de Internet das Coisas, suas diversas aplicações e topologias, assim como algumas propostas de arquitetura para sistemas IoT (*Internet of Things*). Em seguida, uma discussão sobre o uso da plataforma Arduino como hardware em uma topologia IoT é apresentada. Na sequência, são discutidos os protocolos de comunicação utilizados no protótipo, e, por fim, são analisadas a arquitetura e a linguagem de software utilizada no desenvolvimento do aplicativo móvel.

2.2 TECNOLOGIAS IOT

2.2.1 Arquitetura de Sistemas IoT

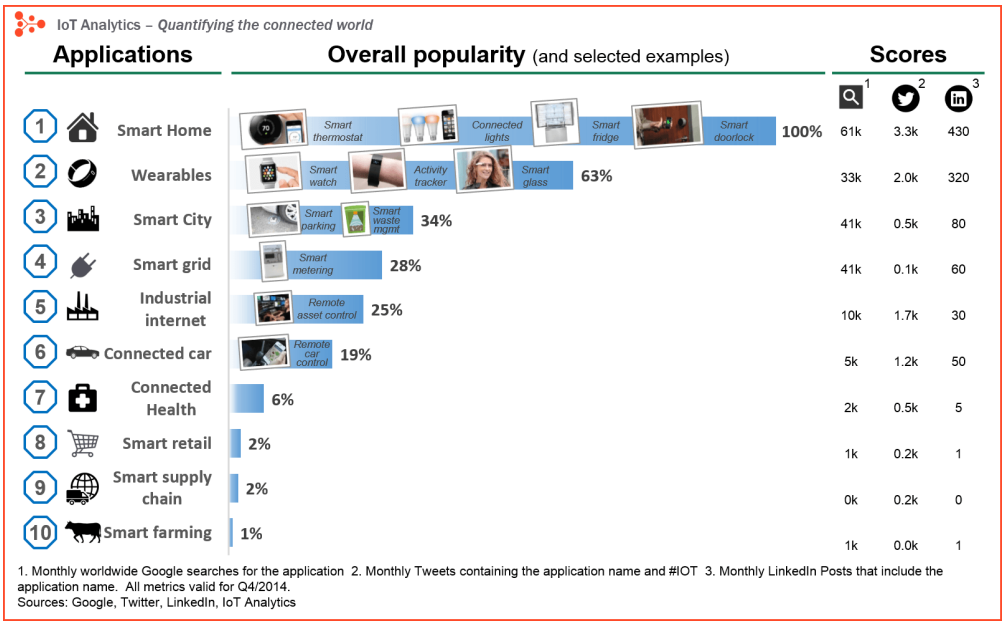
A Internet das Coisas é um conceito que envolve a adição de conectividade e eletrônica a objetos comuns. Acredita-se que, eventualmente, qualquer coisa que possa ser conectada à Internet, será. Imagina-se coisas como eletrodomésticos, roupas, relógios, carros, encomendas, façam parte de uma rede que constantemente colete dados através de sensores e, em alguns casos, tenham atuadores acionados remotamente. Segundo um estudo publicado por GARTNER (2017), 8,4 bilhões de dispositivos já estão conectados à Internet, e para 2020 estão previstos 20 bilhões. Este processo tem sido impulsionado pela redução do custo e pela abundância de eletrônicos com capacidade de conectividade Wi-Fi e Bluetooth. As possibilidades são imensas e os impactos a longo prazo da onipresença do hardware e da Internet são imprevisíveis. KARIMI e ATKINSON (2013, p. 2) listaram algumas aplicações e casos de uso da IoT (*Internet of Things*) mais mencionadas na Internet:

- Comunicação Machine-to-machine;
- Comunicação Machine-to-infrastructure;
- Monitoramento remoto de pacientes hospitalares, diagnósticos e entrega de medicamentos;
- Monitoramento contínuo e atualizações remotas de *firmware* para veículos;

- Acompanhamento de itens em processo de entrega;
- Gestão automática de tráfego;
- Segurança e controle remoto;
- Monitoramento e controle ambiental remoto;
- Automação de casas e estabelecimentos industriais;
- Diversas outras aplicações inteligentes em cidades, distribuição d'água; agricultura, prédios, carros, *tags* e criação de animais.

A Figura 1 ilustra as aplicações mais populares da IoT hoje, listadas em função da popularidade em redes sociais e da quantidade buscas no buscador Google.

Figura 1: Popularidade de tecnologias IoT.



Fonte: IOT ANALYTICS (2017)

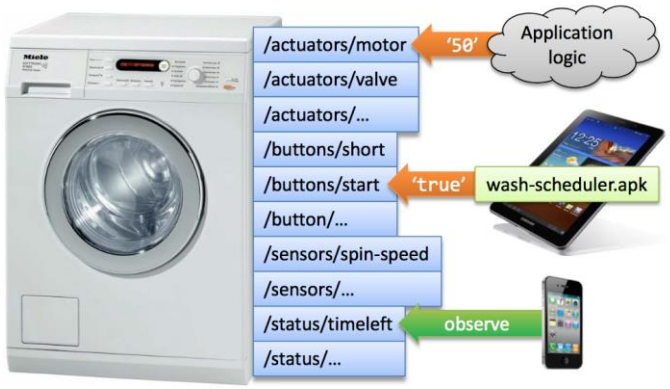
Segundo PIMENTEL (2016), problemas de diversas áreas serão resolvidos na era da IoT, através da descentralização e da onipresença do hardware, do monitoramento e controle em tempo real de objetos de interesse, da automatização de processos, da comunicação constante entre dispositivos, da aquisição e processamento massivo de dados. Cada um dos problemas, no entanto, difere em

seus requisitos e restrições, principalmente no que se refere a questões energéticas, espaciais e de custo.

Apesar dos esforços atuais para padronizar a IoT, é natural que não exista um único padrão universal e que os sistemas tenham arquiteturas diferentes, em função dos requisitos e individualidades de cada projeto. Dispositivos vestíveis, por exemplo, tem extremas restrições quanto à obtenção de energia elétrica e a suas dimensões. Já eletrônicos de automação residencial, como uma lâmpada residencial conectada à Internet, não possuem restrições quanto à alimentação e tem alguma flexibilidade quanto às dimensões. Alguns projetos possuem um hub central de comunicação e armazenamento de dados, outros são completamente distribuídos, usando redes *mesh* com diversos nós para comunicação. A arquitetura será, portanto, consequência das restrições e necessidades do problema a ser solucionado.

KOVATSCH et al. (2012) propuseram uma arquitetura IoT que consiste da migração de toda lógica de aplicação dos dispositivos para um servidor remoto, arquitetura a qual denominaram *Thin Server Architecture*. Os dispositivos deveriam, então, apenas expor uma API (*Application Programming Interface*) de acesso direto aos sensores e atuadores, que seriam mapeados em diferentes requisições. A Figura 2 ilustra um exemplo da arquitetura no caso de uso de uma máquina de lavar roupas conectada à Internet.

Figura 2: Exemplo da Thin Server Architecture.



Fonte: KOVATSCH et. al (2017).

A *Thin Server Architecture* favorece o desacoplamento e diminui a complexidade do código a ser implementado nos dispositivos. Qualquer sensor e atuador da máquina é transparente, ou seja, é acessível via chamadas à API, para usuários autenticados. No entanto, ela requer que, para cada leitura de sensor ou ativação de atuador, uma requisição seja enviada ao dispositivo e devolvida ao remetente. Isto requer que a máquina esteja sempre conectada à Internet, restrição que viria a limitar suas funcionalidades em caso de queda da conexão à Internet, assim como a aumentar seu consumo energético a ponto de inviabilizar o funcionamento do dispositivo se não conectado a uma fonte de alimentação constante. Em vista disso, a implementação pura desta arquitetura não é possível num dispositivo portátil alimentado a bateria, mas alguns de seus conceitos, como a migração da lógica para a nuvem, ainda podem ser aproveitados nesta situação.

Tendo em vista as limitações energéticas de dispositivos portáteis e buscando uma Internet das coisas mais ecológica e sustentável, ABEDIN et. al (2015) propuseram um modelo de sistema energeticamente eficiente para uma “Green-IoT”. Um servidor web armazena representações dos dispositivos em seu banco de dados e funciona como ponte de comunicação entre usuários e seus dispositivos, através de uma API. O diferencial desta arquitetura é a possibilidade de o dispositivo assumir três estados diferentes: *on duty*, *pre-off duty*, e *off duty*, com consumo alto, médio e baixo de energia, respectivamente. O estado *off duty* divide-se em três sub-estados: *hibernate*, *sleep* e *power off*.

No estado de hibernação, o dispositivo apenas faz a leitura seus sensores, não há transmissão ou recebimento de dados. Somente um input específico de algum sensor pode trazê-lo novamente ao estado ativo. Nesta situação, o consumo de energia é baixíssimo. Dispositivos que dependem de fontes renováveis de energia tem, neste momento, a oportunidade de carregar suas baterias para aumentar a expectativa de duração de energia. No estado *sleep*, o dispositivo executa as funções normais do estado ativo (*on-duty*) e retorna ao anterior. No estado *power-off*, o dispositivo está completamente desligado, com consumo nulo de energia.

Neste trabalho, algumas implementações foram inspiradas pelas propostas de arquiteturas citadas acima, como a implementação de partes da lógica de aplicação na nuvem e o uso de estados de economia de energia, com o intuito de estender a duração da bateria. Além disso, como será mostrado posteriormente, soluções novas foram criadas, como a detecção de presença do usuário para ativação do modo de economia de energia máximo.

2.2.2 Arduino como SoC (*System on a Chip*) da Internet das Coisas

O Arduino é uma plataforma *open-source* de eletrônicos, baseada em um conjunto de hardware e software, desenvolvida com o foco na facilidade de uso. Os microcontroladores da família Arduino podem ser conectados a periféricos para, por exemplo, fazer leituras de sensores de aceleração, detectar a ativação de botões, e usar estas entradas para tomar ações como acender uma luz ou fazer uma publicação em uma rede social. O software pode ser desenvolvido na linguagem de programação própria da plataforma, a Arduino Programming Language, ou em qualquer linguagem de programação,

Na plataforma Arduino, boa parte das dificuldades de trabalhar-se com microcontroladores são abstraídas, com o objetivo de torná-la acessível a iniciantes. Isto permite, por exemplo, que professores criem pequenos projetos multidisciplinares em sala de aula, que alunos iniciem estudos de robótica e que startups possam desenvolver protótipos rapidamente. A plataforma mantém, no entanto, as funcionalidades de baixo nível disponíveis para o desenvolvimento de projetos mais avançados.

HRIBERNIK et al. (2017) propuseram um processo de criação de produtos conectados à Internet baseado na plataforma Arduino e elencaram suas vantagens e desvantagens, como detalhado nas Tabelas 1 e 2.

Tabela 1: Vantagens da Plataforma Arduino.

Fator	**Descrição**
Facilidade no desenvolvimento do software	A Arduino Programming Language é tão simples, que até usuários com conhecimentos rudimentares de programação podem começar a usá-la e produzir resultados rapidamente.
Facilidade no desenvolvimento do hardware	O Arduino é fácil de compreender até para usuários sem conhecimento de eletrônica ou de desenvolvimento de hardware.
Esforços da comunidade *open-source*	O Arduino tem uma comunidade muito ativa, que fornece documentação detalhada. Os entusiastas costumam receber os novatos muito bem.
Documentação de Software	Numerosas fontes de documentação disponíveis. Guias de início rápido, blogs, fóruns, e *sites* de tutoriais estão disponíveis para todos.
Preço	O software é *open-source*, portanto é gratuito. As placas e os componentes têm um preço bem baixo.

Fonte: HRIBERNIK et al. (2011).

Tabela 2: Desvantagens da Plataforma Arduino.

Fator	Descrição
Falta de documentação dos componentes	Alguns componentes não possuem documentação adequada, às vezes sequer a possuem, consequentemente não poderiam ser usados em protótipo ação rápida. Isto impacta diretamente na habilidade do usuário de implementar suas ideias rapidamente.
Documentação do componente muito técnica	A documentação de alguns componentes é de natureza altamente técnica, útil somente para pessoas experientes. Isto também impacta diretamente na habilidade do usuário de implementar suas ideias rapidamente.
Falta de exemplos de uso de componentes	A falta de exemplos simples é uma dificuldade para usuários com pouco conhecimento técnico.
Especificações imprecisas de componentes	Especificações imprecisas de alguns componentes fizeram com que eles se comportassem diferentemente do esperado. Isto cria frustrações no processo de desenvolvimento porque a implementação de uma ideia se comporta diferentemente do esperado sem razão aparente.

Fonte: HRIBERNIK et al. (2011).

2.2.3 Comunicação de Curta Distância: Bluetooth

O Bluetooth é o padrão de comunicação sem fio de curta distância mais difundido da atualidade, segundo BLUETOOTH SIG (2018). Concebido originalmente como uma alternativa aos cabos seriais RS-232, atualmente ele está presente em

virtualmente todos os *smartphones* do mundo. Hoje, ele dá suporte a muitas outras funcionalidades, como envio de arquivos, transmissão de áudio, pontos pessoais de acesso à Internet e comunicação com sensores. Criado em 1994 pela Ericsson, atualmente está sob controle do Bluetooth SIG (*Special Interest Group*), uma associação de mais 30.000 empresas focadas na padronização e na evolução do Bluetooth, que já está em sua especificação 5.0.

Como descrito por ROTH (2013), a tecnologia opera na banda de 2.400 a 2.4835 GHz, que é uma faixa ISM (*Industrial, Scientific and Medical*) de uso geral, de aceitação mundial. Para reduzir a interferência causada por outros dispositivos operando na mesma banda, o Bluetooth usa a técnica de *frequency-hopping spread spectrum*, que consiste na mudança constante de frequência de portadora, alternando o canal de comunicação após um período definido. No Bluetooth clássico, anterior à especificação 4.0, são 79 canais de 1 MHz disponíveis para a transmissão, alternando entre eles aproximadamente 800 vezes por segundo. As modulações disponíveis são a π/4 DQPSK (*Differential Quadrature Phase Shift Keying*) e a 8 DPSK (*Differential Phase Shift Keying*), dependendo da qualidade do sinal.

Em 2010, o Bluetooth SIG finalizou a especificação 4.0, disponível em BLUETOOTH SIG (2010), que contém a tecnologia Bluetooth *Low Energy*, criada para dar suporte a um leque de novas aplicações que requerem baixo consumo de energia, mantendo o alcance das especificações anteriores.

O BLE (Bluetooth Low Energy) mantém a topologia do Bluetooth clássico, que divide os dispositivos em *master* e *slave* (ou centrais e periféricos), de acordo com seus papéis. Para padronizar a comunicação entre eles, foram introduzidos os atributos genéricos de perfil (GATT), que listam identificadores padronizados para funcionalidades como monitoramento de glicose corporal, navegação, estado da carga de bateria, entre outros, facilitando assim a comunicação de dispositivos IoT de fabricantes diferentes.

As características introduzidas pelo Bluetooth *Low Energy*, unidas a sua popularidade e difusão mundial, fazem com que ele emerja como uma forte alternativa de comunicação direta entre dispositivos pertencentes à Internet das Coisas. Portanto, foi escolhido para objeto de estudo deste trabalho.

2.2.4 Comunicação de Longa Distância: GPRS

Como descrito por CAI e GOODMAN (1997, p. 122), o General Packet Radio *Service* (GPRS) é um serviço móvel de acesso à Internet construído sobre a rede celular digital de segunda geração, a GSM (Global System for Mobile Communications). A tecnologia foi criada com o objetivo de substituir o sistema de transferência de dados disponível no início dos anos 90, o *Circuit Switched Data*, que, como constatado por KORHONEN (2001), obtém uma taxa de transferência média de meros 9.6 kbit/s, usando técnicas de chaveamento sobre a rede GSM. Por ser uma evolução sobre o GSM original, não uma nova geração da tecnologia, o GPRS também é chamado de 2.5G. Duas décadas após a sua criação, a tecnologia já foi superada por tecnologias como a HSPA (*High Speed Packet Access*), também conhecida como 3G, e LTE (*Long Term Evolution*), conhecida como 4G, melhores em taxa de transferência e latência.

O GPRS é um serviço de capacidade sob demanda, conforme CAI e GOODMAN (1997, p. 125). Isso significa que a taxa de transferência em uma sessão varia em função da quantidade de usuários conectados à torre e da quantidade de pacotes a serem transferidos. Usando multiplexação, o GPRS permite que centenas de usuários se conectem a uma torre simultaneamente. As taxas de transferência variam também com a qualidade do sinal e sua codificação correspondente. A Tabela 3 relaciona os esquemas de codificação com as respectivas velocidades máximas teóricas de transferência.

Tabela 3: Taxa de transferência do GPRS em função da codificação.

Esquema de Codificação	Taxa de Codificação	Taxa de Transferência Máxima
CS-1	1/2	8,0 kbps
CS-2	≅ 2/3	12,0 kbp
CS-3	≅ 3/4	14,4 kbps
CS-4	1	20,0 kbps

Fonte: CAI e GOODMAN (1997, p. 125).

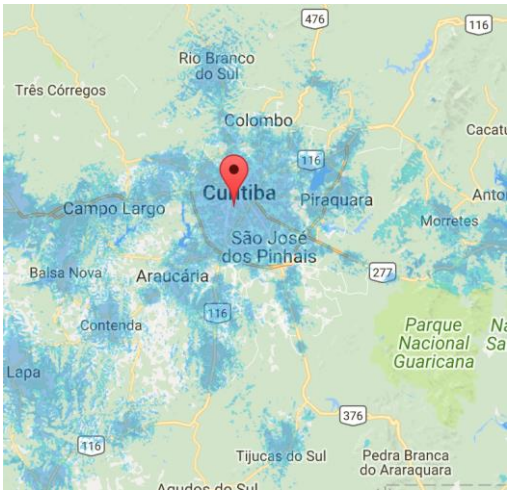
Na Figura 3 é apresentada uma medição experimental de velocidade e latência da comunicação GPRS da operadora TIM. O teste foi realizado na cidade de Pinhais - PR.

Figura 3: Medição de velocidade e latência GPRS.

Fonte: Os autores.

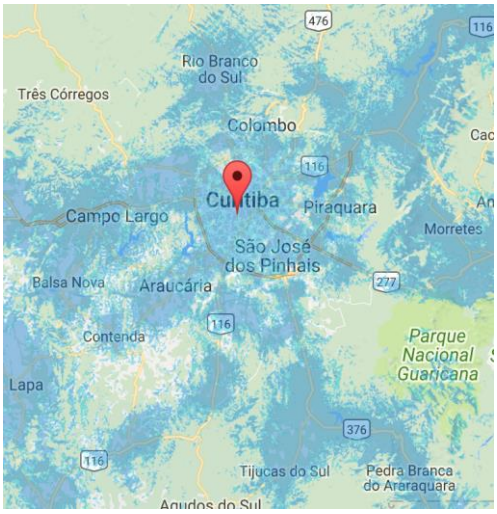
De acordo com o *website* da operadora de telecomunicações TIM (2017), a rede GSM, que dá suporte à tecnologia GPRS e HSPA, ainda mantém área de cobertura visivelmente superior à da rede LTE no Brasil. As Figuras 4 e 5 apresentam respectivamente a cobertura LTE e GSM, em azul, da operadora de telecomunicações TIM.

Figura 4: Cobertura LTE da TIM na Região Metropolitana de Curitiba.



Fonte: TIM (2017).

Figura 5: Cobertura GSM da TIM na Região Metropolitana de Curitiba.



Fonte: TIM (2017).

Apesar de não ser a tecnologia mais recente de comunicações sem fio, o GPRS foi escolhido para a execução deste projeto pela sua vasta cobertura, baixo custo dos módulos de comunicação e taxa de transferência satisfatória aos requisitos do projeto (que envia requisições com tamanhos da ordem de poucos *kilobytes*). A latência, de poucos segundos, também satisfaz os requisitos do projeto.

2.3 DESENVOLVIMENTO DE APLICAÇÕES MÓVEIS

2.3.1 Padrão de Arquitetura

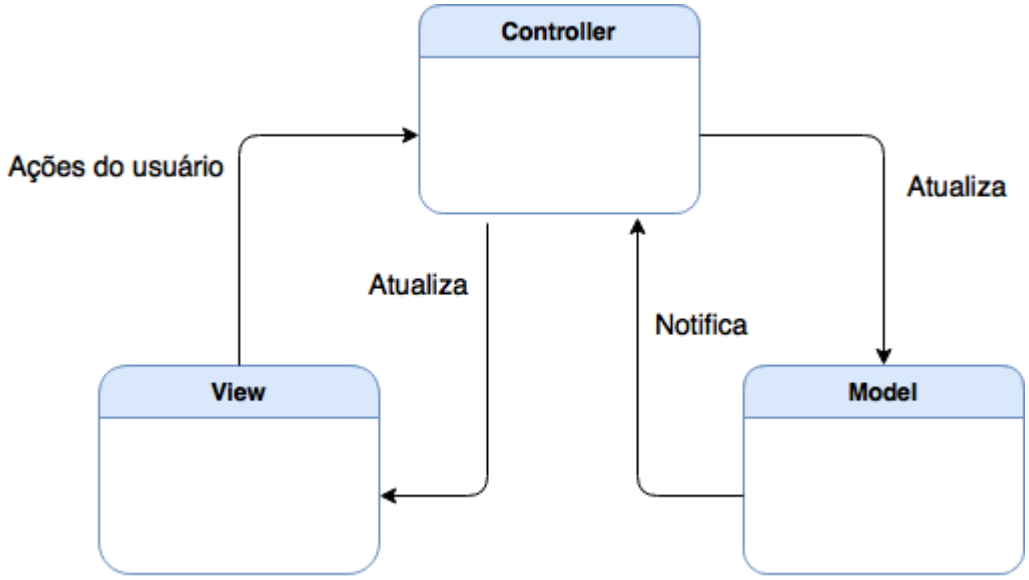
O Model-View-Controller é um padrão tradicionalmente usado no desenvolvimento de software que apresenta interface gráfica de usuário. Usado primeiramente em aplicações desktop, o padrão propagou-se para o desenvolvimento de aplicações web e móvel.

Como descrito por POPE (1998), o Model-View-Controller promove o desacoplamento e o reuso de código, através da separação entre elementos a serem renderizados na tela, elementos de dados puros e a lógica de aplicação. O objetivo é isolar representações internas de informação das maneiras pelas quais esta informação é apresentada ao usuário. O padrão possui três partes principais, o *Model*, a *View* e o *Controller*:

- *Model* é onde os dados residem. Dados persistentes, objetos que modelam dados, e código de download de itens residem aqui;
- *View* é a camada referente aos itens visíveis do aplicativo. Como estes itens não possuem lógica, podem ser reusados por toda aplicação; e
- *Controller* é a parte do código que intermedia a comunicação entre *View* e *Model*.

O diagrama da Figura 6 representa os três elementos do Model-View-Controller e suas relações.

Figura 6: Model-View-Controller.



Fonte: Os autores.

A *View* repassa as ações do usuário, como eventos em toques de botões, ou seleções de itens em listas, para o *Controller*. O *Controller*, por sua vez, atualiza os elementos na *Model*, que pode ser qualquer estrutura de armazenamento de dados, desde um registrador a um banco de dados. Após isso, o *Controller* atualiza a *View* para o estado condizente ao estado atual da aplicação e com as informações presentes na *Model.*

LEFF e RAYFIELD (2001) pontuaram os motivos pelos quais o MVC torna mais fácil desenvolver e manter aplicações modernas:

- A aparência das aplicações pode ser mudada drasticamente sem que as estruturas de dados e as lógicas de negócio precisem ser modificadas;
- As aplicações podem manter interfaces diferentes, o que torna a implementação de funcionalidades, como o suporte a múltiplas línguas, prático.

Devido às qualidades mencionadas, o Model-View-Controller foi escolhido como padrão de arquitetura para nortear a codificação do aplicativo deste projeto.

2.3.2 Linguagem de Programação Swift

Swift é uma linguagem de programação de propósito geral criada pela Apple Inc. Lançada em 2014, seu objetivo é substituir o Objective-C como linguagem padrão de seus sistemas operacionais, segundo APPLE INC. (2014), mantendo, no entanto, o suporte a ambas. A escolha do Swift para este projeto deve-se a suas três principais qualidades: sintaxe limpa e moderna, resiliência contra códigos errôneos e performance.

A linguagem é considerada multi-paradigma pois, além de seguir a orientação a objetos, também pode ser orientada a protocolos, funcional, imperativa e estruturada em blocos. Em 2015, a Apple tornou o Swift *open-source*, criou o portal SWIFT.ORG (2018) e hospedou o código fonte no serviço GitHub, possibilitando a qualquer indivíduo colaborar com novas proposições para a evolução da linguagem.

Sua sintaxe, enxuta e legível, lembra linguagens modernas difundidas de *scripting* como Ruby ou Python. Ela possui qualidades como a ausência de ponto e vírgula no término de uma linha e a facilidade para se declarar variáveis, cujos tipos não precisam ser explicitados pois são inferidos. O Swift também se deixou inspirar por funcionalidades presentes em mais outras linguagens, como:

- A sintaxe de *getter / setter* do C#;
- A declaração de variáveis seguida de dois pontos e seu tipo, como em Pascal;
- Interpolação de *string*, como na maioria das linguagens script;
- O ponto de interrogação após um tipo para denotar que a variável pode ser nulificada, como em Ceylon;
- Os operadores “..<” e “…” usados em Ruby.

Um mérito adicional do Swift é a resiliência contra código errôneo, já que as seguintes medidas de precaução são tomadas:

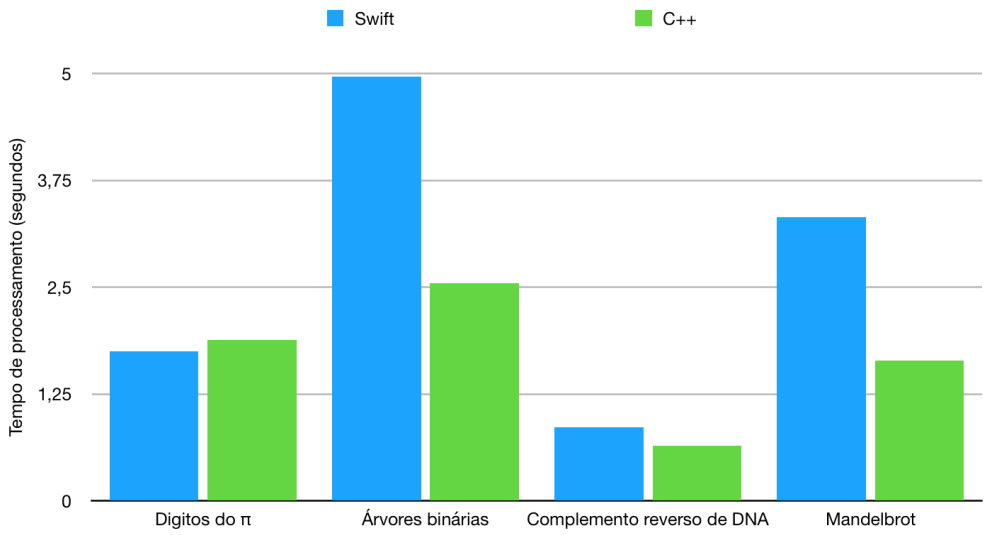
- Verificação de inicialização de variáveis antes do uso;
- Índices de vetores são verificados contra erros de acesso fora dos limites;

- Controle do overflow de inteiros;
- Declaração de variáveis que podem ser nulas como *optionals*;
- Gestão de memória automática em tempo de compilação;
- Gestão de erros que permite recuperação após falhas inesperadas.

Tais recursos viabilizam o desenvolvimento de aplicativos móveis para iOS com qualidade, o que vai ao encontro da necessidade deste projeto em ter um aplicativo robusto, o qual não terá falhas durante o rastreamento de um objeto.

Sua performance está acima da de linguagens *scripting high-level*, já que seu código é compilado para código de máquina pelo compilador LLVM (*Low Level Virtual Machine*), contribuindo para que ela tenha uma performance similar à da linguagem C++. Na Figura 7, apresenta-se uma comparação de performance entre as duas linguagens em alguns algoritmos.

Figura 7: Comparativo de performance entre Swift e C++.



Fonte: (GOUY,2017).

Em comparação ao seu antecessor, o Objective-C, o Swift manteve funcionalidades como a ligação dinâmica (*dynamic dispatch*), que viabiliza o polimorfismo, a ligação tardia (*late binding*), que permite que métodos de classes sejam

buscados em tempo de execução. Outra funcionalidade mantida é a programação extensível (*extensible programming*), que permite adicionar métodos a classes já existentes, independente da disponibilidade do código fonte.

Dentre as mudanças de sintaxe, destaca-se o abandono da chamada de função usando colchetes, herdada da linguagem Smalltalk, que foi substituída pela *dot-syntax*, o mesmo padrão de linguagens como Java e C#, de grande adoção.

No MacOS e no iOS, o código Swift pode ser mesclado ao código Objective-C, sendo necessárias algumas pequenas adaptações e a declaração das variáveis que se deseja acessar usando as duas linguagens em um arquivo chamado *bridging header*.

As características do Swift viabilizam o desenvolvimento de aplicativos móveis mais robustos e ágeis para iOS, o que vai ao encontro das necessidades deste projeto, que visa oferecer um aplicativo estável e responsivo para o usuário final. Além disso, as qualidades da sintaxe permitem uma codificação mais enxuta e legível, e favorecem o reuso de código, tornando a codificação mais prática e rápida.

2.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou conceitos ligados aos tópicos mais relevantes para o desenvolvimento deste trabalho, englobando tecnologias de IoT e técnicas de desenvolvimento de aplicações móveis. Partindo destes conceitos, o próximo capítulo apresentará a solução criada e desenvolvida neste trabalho, a qual tem o intuito de ser aplicada ao monitoramento de objetos pessoais.

3 POCKET GUARD

3.1 CONSIDERAÇÕES INICIAIS

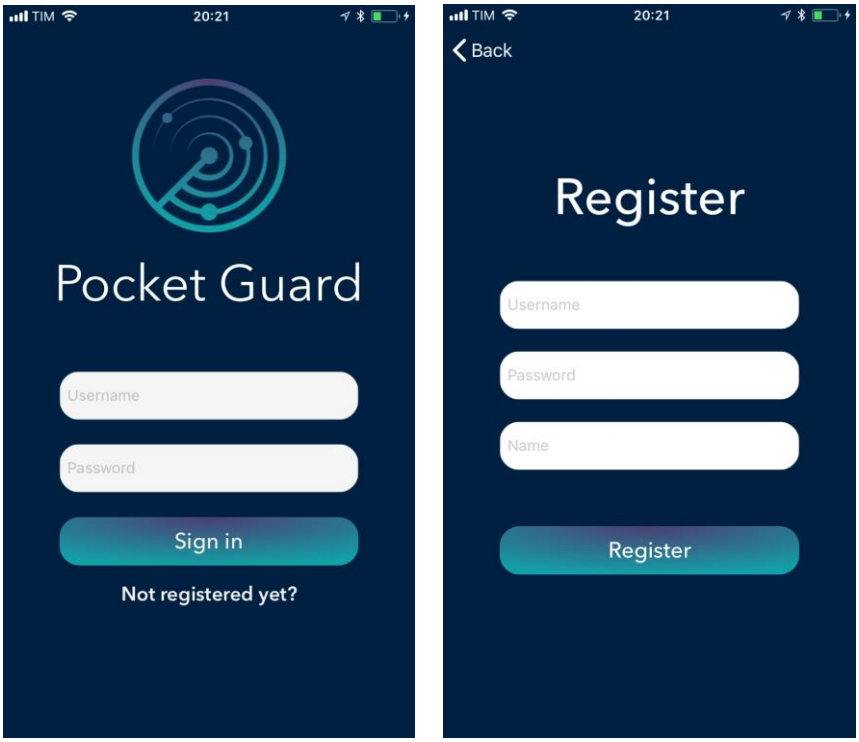
Partindo dos fundamentos teóricos anteriormente apresentados, neste capítulo serão discutidos o funcionamento e a operação do sistema desenvolvido para este trabalho. Inicialmente é feita uma descrição do que é esperado do sistema, em seguida uma visão geral do seu funcionamento e, por fim, uma análise do desenvolvimento de cada parte do projeto.

3.2 DESCRIÇÃO DO SISTEMA

O Pocket Guard é um sistema de rastreamento composto por um dispositivo portátil conectado à Internet, um servidor *web* e um aplicativo para *smartphone*. O protótipo, um microcontrolador Arduino, pode ser posicionado pelo usuário em qualquer pertence, como uma bolsa ou mochila, em seu automóvel, nas gavetas de seus móveis e onde mais desejar. Após ativado, o dispositivo liga seu acelerômetro e, caso detecte um padrão atípico de movimentação, correspondente a um furto, notifica o usuário imediatamente. Tal notificação é mostrada na tela do *smartphone* do usuário, acompanhada de uma indicação sonora. Ao abri-la, o usuário é levado ao aplicativo Pocket Guard, no qual pode acompanhar em tempo real a movimentação e a localização do seu rastreador.

Para fazer uso do sistema, o usuário precisa primeiramente fazer o download do aplicativo Pocket Guard em seu *smartphone* e possuir um dispositivo portátil de rastreamento Pocket Guard com um chip GSM ativo inserido. Não há botões ou telas que apresentem informações diretamente no rastreador, sendo, portanto, todo o monitoramento e configuração do dispositivo realizados por meio do aplicativo Pocket Guard. Ao abrir o aplicativo, a tela de cadastro é apresentada ao usuário, conduzindo-o de modo que ele cadastre seu nome, crie um nome de usuário e defina uma senha. A Figura 8 apresenta as telas de *login* e de cadastro de usuários.

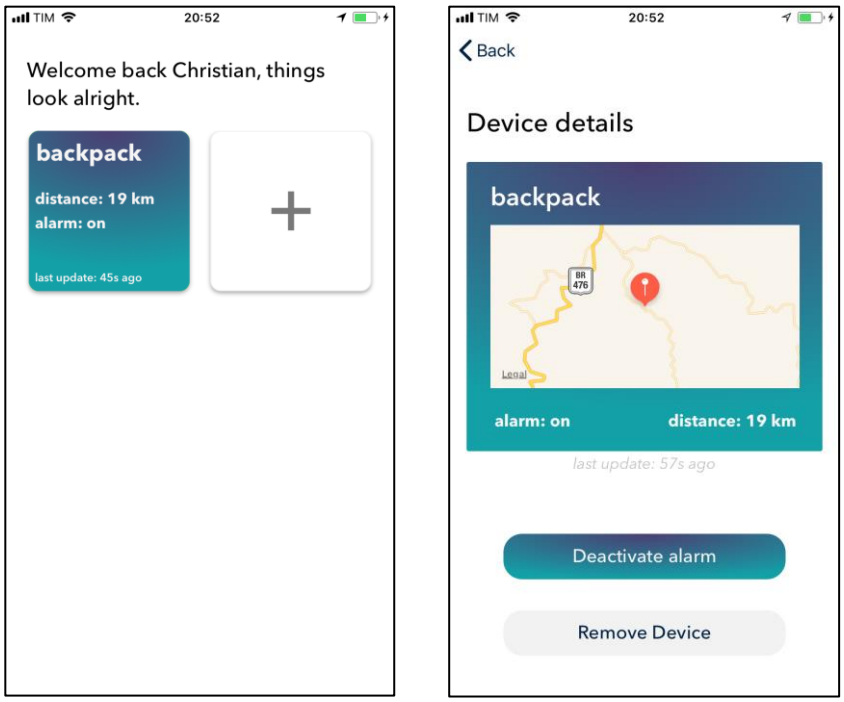
Figura 8: Telas de *login* e cadastro do Pocket Guard.



Fonte: Os autores.

Após seu cadastro, o usuário pode fazer o *login* e parear seu primeiro dispositivo. Para executar o pareamento, basta que o usuário ligue seu rastreador Pocket Guard, abra a tela de pareamento em seu celular e clique no botão *pair*. Após isso, o usuário deve apenas posicionar o dispositivo no local de interesse. A partir deste momento, o sistema estará ativo e passará a monitorar os sinais de movimentação do acelerômetro, sua posição geográfica via GNSS e distância em relação ao *smartphone* do usuário via Bluetooth, enviando relatórios ao servidor com um intervalo predeterminado de 5 minutos. A Figura 9 apresenta a tela principal do aplicativo com um dispositivo cadastrado, assim como a tela de detalhamento deste.

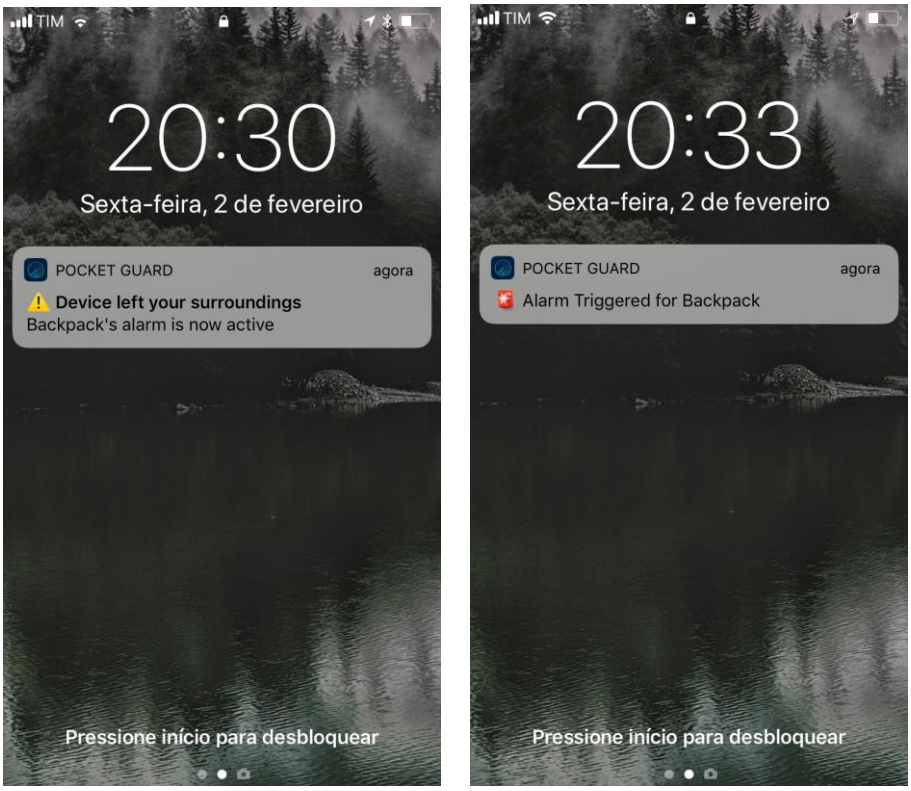
Figura 9: Telas de listagem de rastreadores e de detalhes de dispositivo.



Fonte: Os autores.

Assim que o dispositivo se encontra distante do usuário, um alerta é enviado para seu celular, avisando que ele já não está mais em sua proximidade. A partir deste momento, caso seja detectada alguma movimentação do rastreador, característica de um furto, o dispositivo entrará em modo de disparo de alarme. Uma notificação é enviada imediatamente ao usuário, assim como atualizações constantes de posição geográfica. A Figura 10 contém capturas de tela com as notificações mencionadas.

Figura 10: Notificações de saída de proximidade e de disparo de alarme.



Fonte: Os autores.

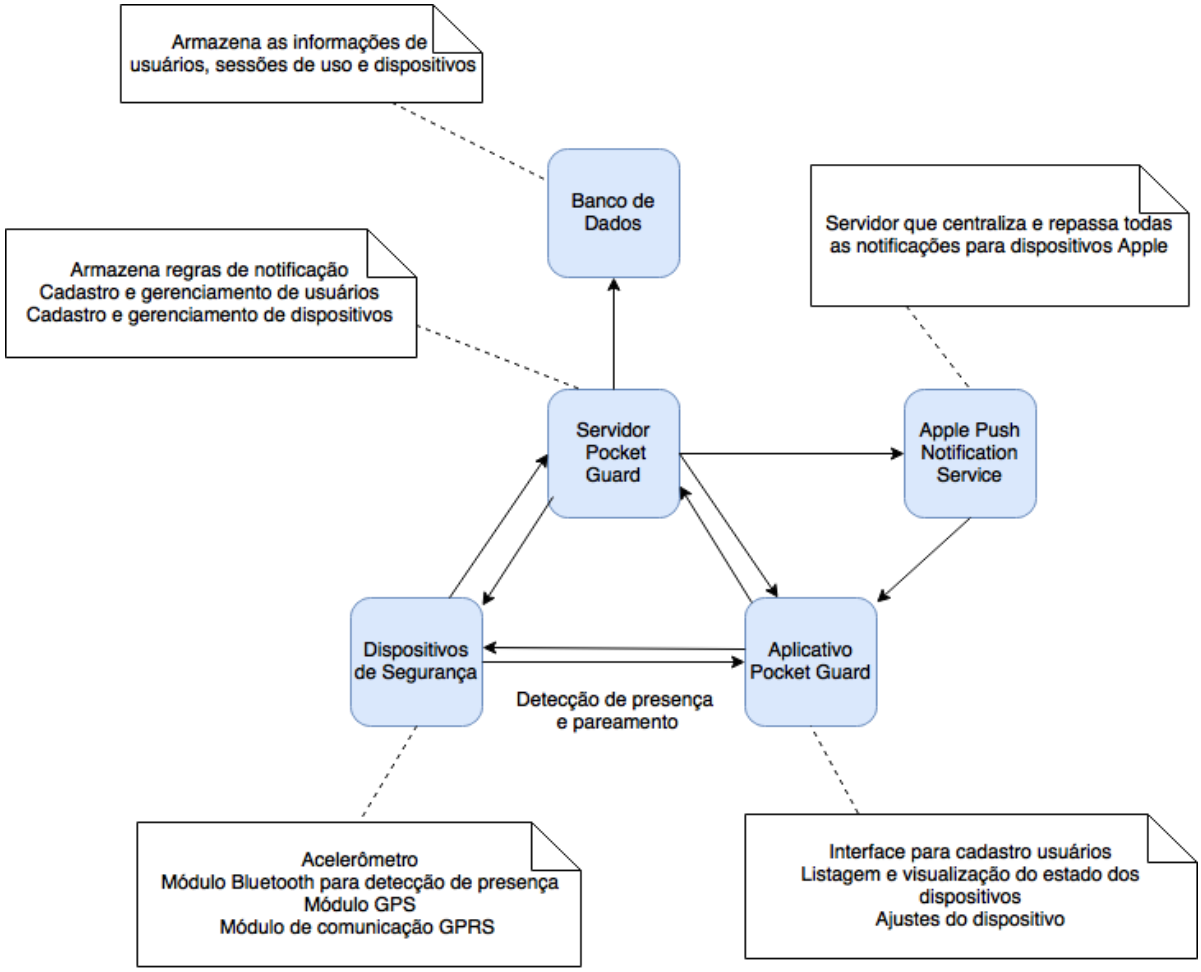
O usuário, ao deslizar o dedo sobre a notificação recebida, é levado diretamente ao aplicativo Pocket Guard, onde pode consultar a localização do dispositivo e desativar o disparo do alarme se desejar.

Imagina-se que os dispositivos de segurança, os rastreadores Pocket Guard, possam ser posicionados em uma infinidade de locais. Por exemplo, pode ser posicionado dentro de mochilas ou bolsas, para que o usuário seja notificado caso se afaste destes pertences; pode ser deixado em automóveis que serão estacionados na rua ou entregues a terceiros, como em um estacionamento ou serviço de *vallet*; pode ser guardado em uma gaveta para que o dono seja notificado quando a abrirem; pode ser acoplado a portas ou portões para enviar alertas de invasões; entre muitas outras aplicações e possibilidades.

O aplicativo Pocket Guard e o dispositivo rastreador comunicam-se por Bluetooth para realizar o pareamento entre o usuário e o rastreador, assim como para

detecção de distância entre eles, assumindo-se sempre que o dono do rastreador esteja com seu *smartphone*. Após a configuração do rastreador, ele comunica-se periodicamente com o servidor web enviando requisições HTTP (*Hypertext Transfer Protocol*) com atualizações de status. Além disso, existe a comunicação entre o aplicativo e o servidor, também através de requisições HTTP, para consultar a situação dos dispositivos, cadastrar usuários e iniciar suas sessões. A Figura 11 apresenta o diagrama de blocos do sistema.

Figura 11: Diagrama geral do sistema *Pocket Guard*.



Fonte: Os autores.

As principais funcionalidades do sistema Pocket Guard são a detecção de movimento do dispositivo rastreador por meio de acelerômetro; a comunicação do

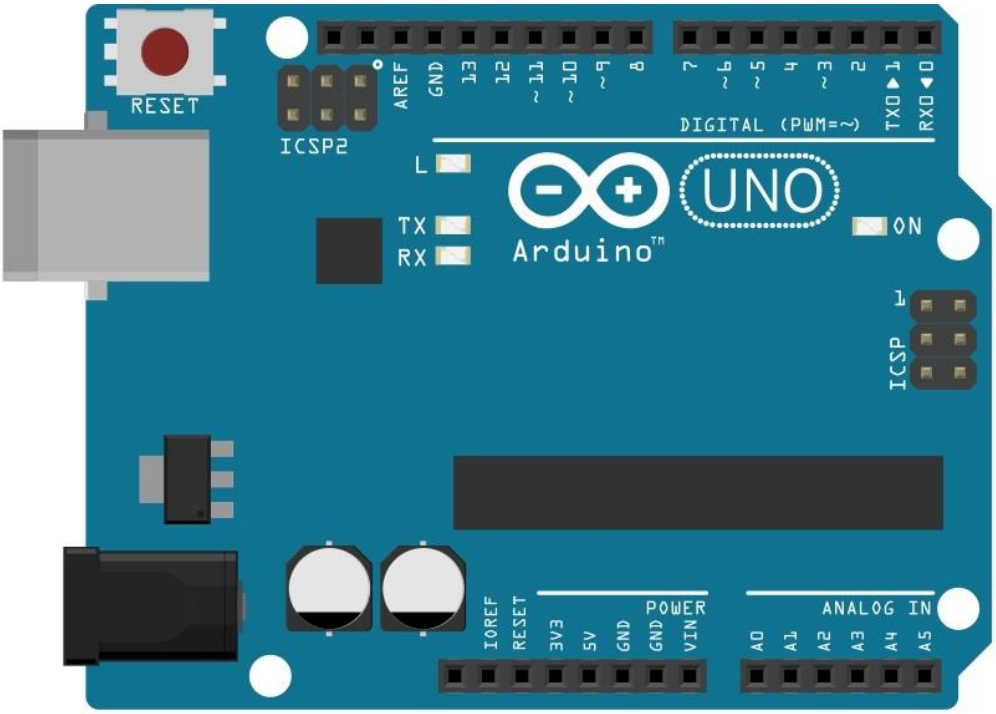
dispositivo com o servidor através de rede GPRS; o rastreamento do dispositivo através do sistema GNSS; um sistema de cadastro e *login* para um usuário poder monitorar vários dispositivos; e a detecção de proximidade entre o *smartphone* e o dispositivo rastreador, pareados via Bluetooth, para evitar disparos causados pelo próprio usuário.

3.3 DESENVOLVIMENTO DO PROTÓTIPO

3.3.1 Hardware

A placa base utilizada foi a Arduino UNO Rev3, a mais simples e de menor custo da série Arduino. Esta placa é composta de um processador ATmega328, 14 pinos de entrada/saída digital (dos quais 6 podem ser usados como PWM), 6 pinos de entrada analógica, 1 cristal oscilador de 16 MHz, entre outras características.

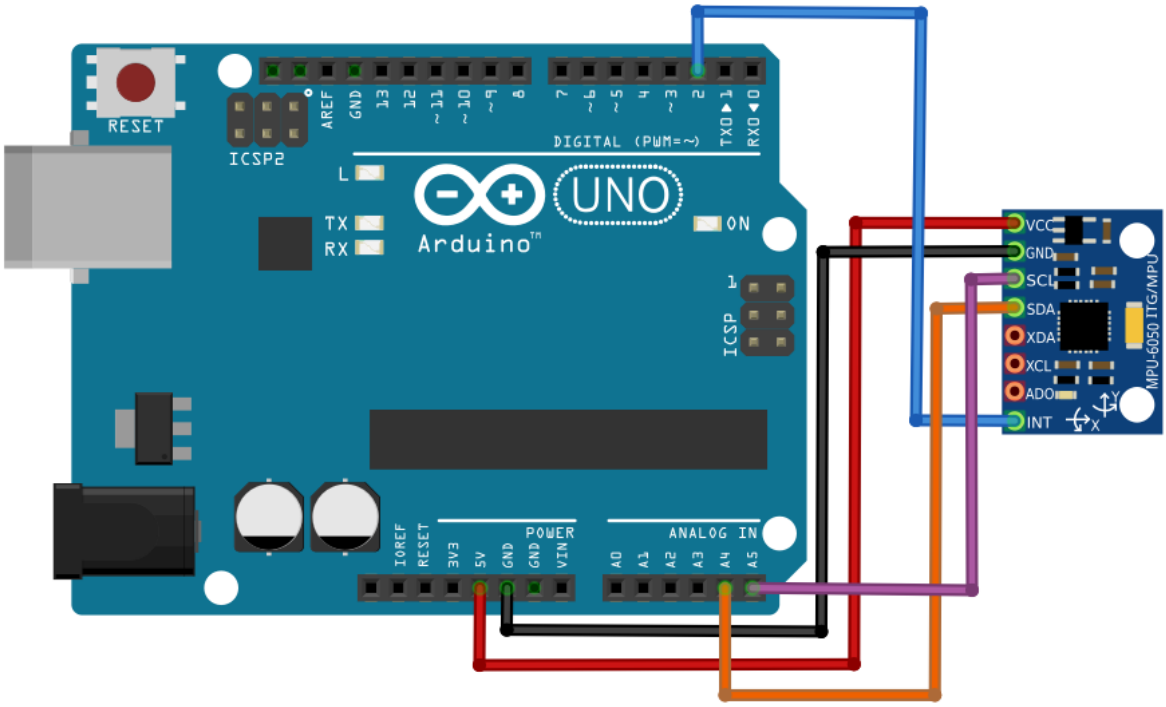
Figura 12: Esquemático da placa Arduino Uno Rev3.



Fonte: Os autores.

Para a função de acelerômetro, utilizou-se o módulo MPU 6050, um dispositivo que contém um acelerômetro de 3 eixos. Possui, também, um giroscópio de 3 eixos que não foi necessário no escopo deste projeto. O módulo é alimentado em 5V e se comunica com o Arduino através do protocolo I2C (*Inter-Integrated Circuit*), de modo que o pino SDA (*Serial Data*, que transmite os dados) é conectado ao pino A4 da placa base e o pino SCL (*Serial Clock*, que sincroniza os dispositivos) é conectado ao pino A5 da placa base. O módulo também faz uso do pino de interrupção externo do Arduino, assim, o pino INT deve ser ligado ao pino 2 (INT0) da placa base.

Figura 13: Esquemático do módulo MPU 6050 conectado ao Arduino Uno.

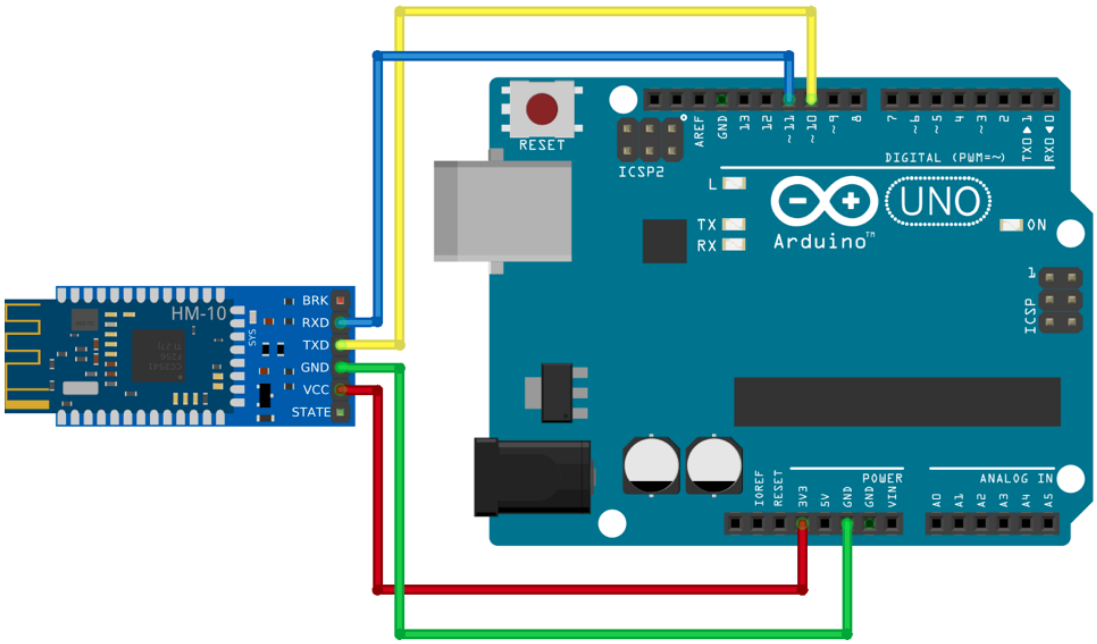


Fonte: Os autores.

Para o Bluetooth, utilizou-se o módulo HM-10 BLE, que se comunica a outros dispositivos utilizando a tecnologia BLE (Bluetooth Low Energy), também chamado de Bluetooth 4.0. O módulo se comunica com a placa base através do protocolo serial. Para isso, foi utilizada a biblioteca SoftwareSerial, disponível já na IDE (*Integrated Development Environment*) do Arduino, de modo que os pinos digitais 10 e 11 da placa

foram transformados em entradas Rx (*Receiver*) e Tx (*Transmitter*) seriais, que se conectam ao Rx e Tx do módulo. Este módulo é alimentado em 3,3 V.

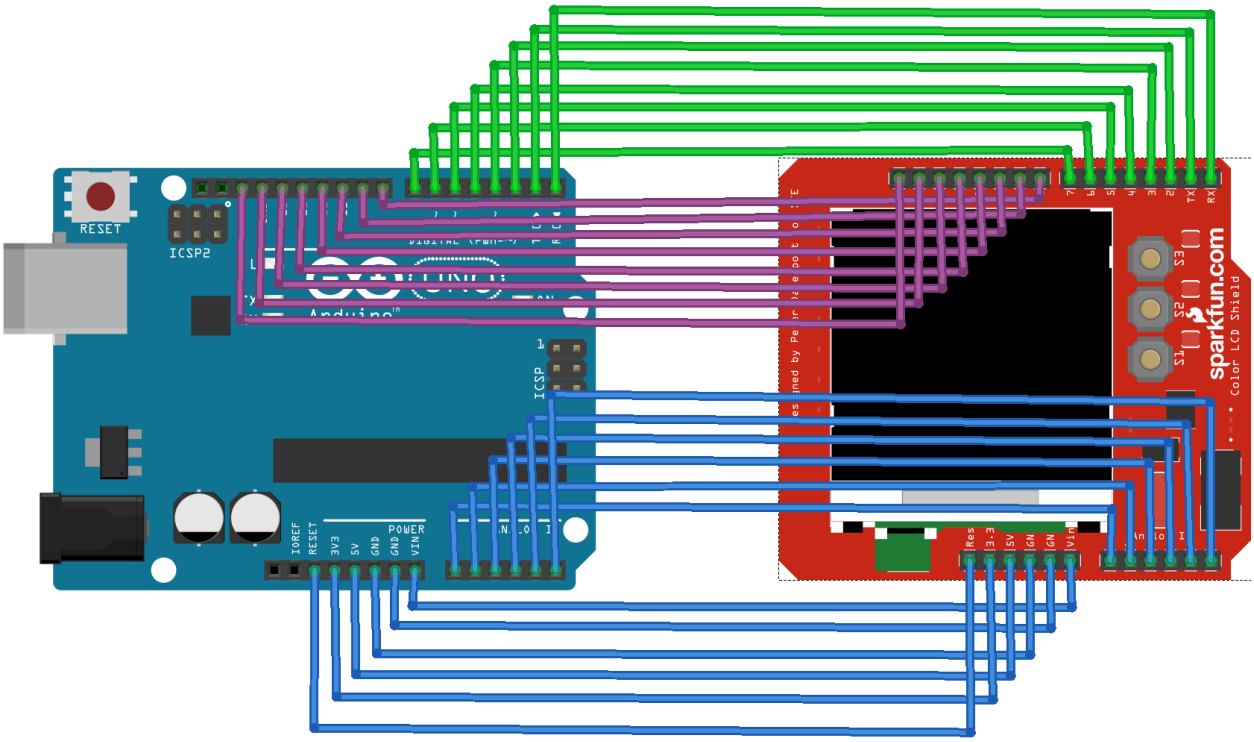
Figura 14: Esquemático do módulo HM-10 conectado ao Arduino Uno.



Fonte: Os autores.

Para o GNSS e o GSM, foi utilizado o módulo SIM 808. Este módulo contém funcionalidades de GNSS, GSM e Bluetooth. No entanto, como o dispositivo Bluetooth deste módulo não possuía os comandos necessários para o projeto, foi decidido ignorar estas funcionalidades e adquirir o já mencionado HM-10 BLE. Este módulo é um *shield*, isto é, ele é conectado a todos os pinos do Arduino, de modo que ele utiliza os pinos que lhe são necessários, e espelha os pinos que não são, para conexão dos demais módulos.

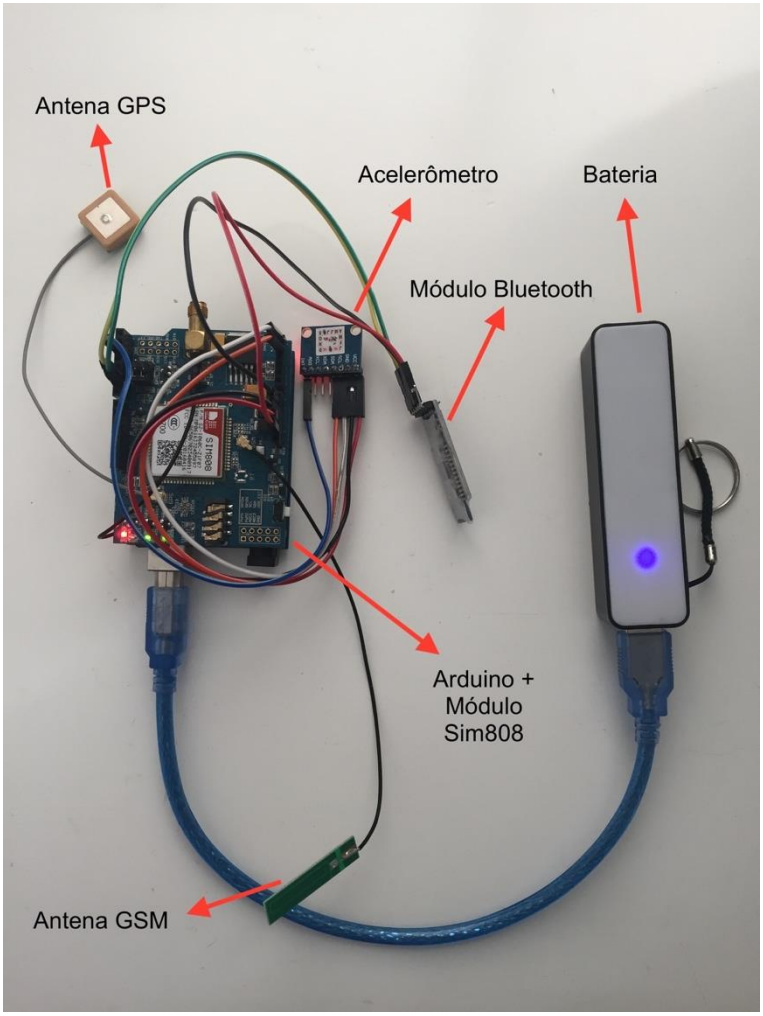
Figura 15: Esquemático do módulo SIM 808 conectado ao Arduino Uno.



Fonte: Os autores.

A Figura 16 apresenta fotografia da composição completa do hardware do rastreador Pocket Guard.

Figura 16: Composição completa do hardware.

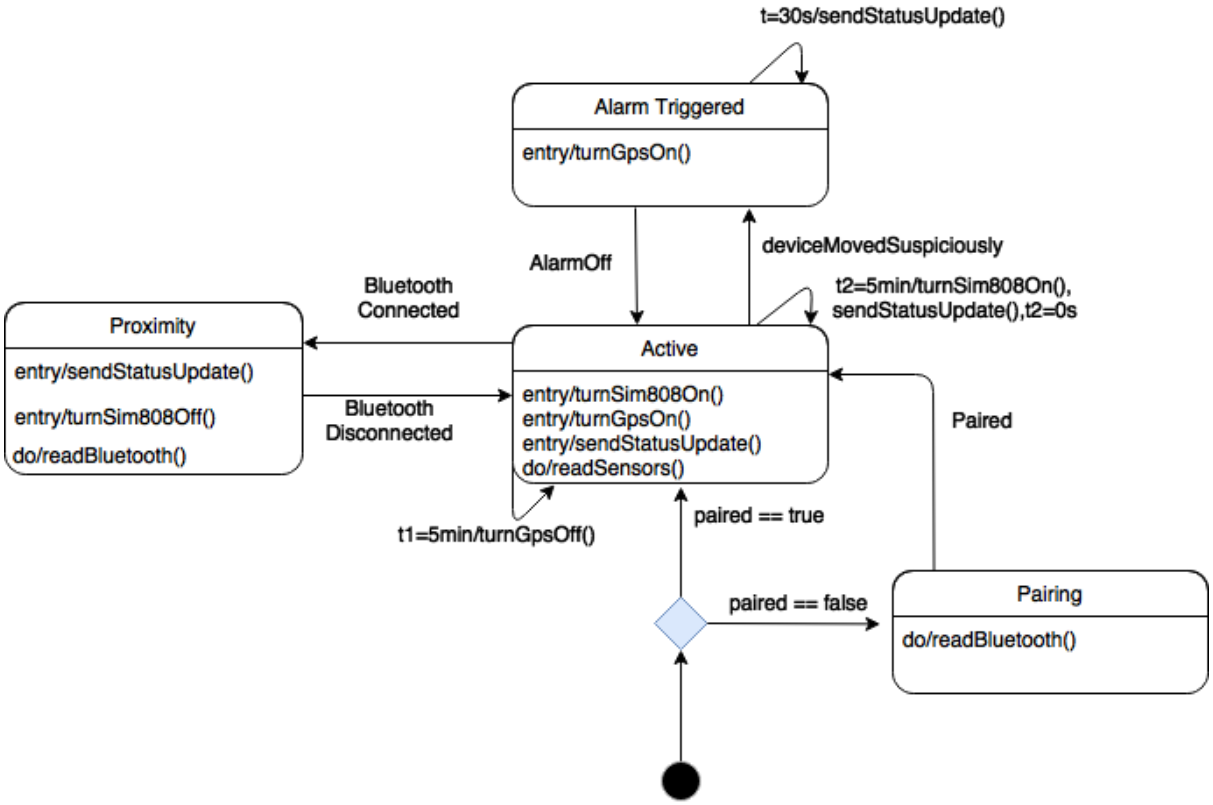


Fonte: Os autores.

3.3.2 Firmware

O sistema funciona utilizando o conceito de máquina de estados. Foram definidos quatro estados: *Active*, *Alarm*, *Proximity* e *Pairing*. Na Figura 17, apresenta-se um diagrama de estados no padrão UML.

Figura 17: Diagrama da máquina de estados implementada no rastreador.



Fonte: Os autores.

No estado *Active* (“ativo”), que é iniciado quando o usuário liga o dispositivo, caso o dispositivo já tenha sido configurado, o sistema fica continuamente checando os dados do acelerômetro. Caso os valores do acelerômetro indiquem um movimento suspeito, o sistema muda o estado para *Alarm Triggered* (“alarme disparado”). Se, antes disso, o dispositivo detectar, através do Bluetooth, que o usuário está próximo, o sistema muda o estado para *Proximity* ("proximidade").

No estado *Alarm Triggered*, o dispositivo para de checar o acelerômetro e o usuário recebe uma notificação no smartphone indicando a movimentação suspeita que ativou este estado. O usuário receberá dados do GNSS indicando a posição atual do dispositivo constantemente e pode desativar o alarme quando quiser, através do aplicativo.

No estado *Proximity*, ativado ao detectar-se que o rastreador está próximo do usuário, o dispositivo para de checar o acelerômetro e desativa o módulo SIM 808, que

gerencia a comunicação GPRS e GNSS. Assim que o dispositivo se distancia do usuário, o dispositivo retorna ao estado Active; o usuário é notificado quanto a isso.

Por fim, o último estado é o *Pairing* (“Pareando”), no qual se encontra o dispositivo que ainda não está pareado com nenhum usuário. Neste estado, o dispositivo fica constantemente varrendo as leituras do módulo de Bluetooth para encontrar algum aparelho próximo que esteja tentando conectar. Ao parear com algum *smartphone*, o dispositivo muda para o estado Active.

Em virtude da existência de apenas um canal de comunicação serial no Arduino Uno e da necessidade de comunicar-se com os módulos Bluetooth e Sim808, assim como enviar informações preciosas de debug para o monitor Serial, foi necessário emular uma comunicação serial através dos pinos de GPIO (*General-purpose input/output*). A biblioteca SoftwareSerial, disponibilizada pelos criadores do Arduino, permite que outros pinos digitais sejam assinalados como pinos de comunicação serial e foi utilizada para resolver o problema de múltiplas comunicações seriais neste projeto.

Outras bibliotecas foram usadas para auxiliar no desenvolvimento, a MPU6050, para realizar as leituras no acelerômetro, e a ArduinoJson, disponibilizada por BLANCHON (2017), para fazer a análise das mensagens enviadas pelo serviço.

Todo o código do *firmware* pode ser encontrado no apêndice C.

3.4 DESENVOLVIMENTO DO APLICATIVO

3.4.1 Ferramentas de Desenvolvimento

Para o desenvolvimento do aplicativo, foi utilizado o ambiente integrado de desenvolvimento XCode, criado pela Apple Inc, em sua versão 9.2. O software é gratuito e está disponível apenas para o sistema operacional macOS. Algumas de suas funcionalidades de destaques são:

- Compatibilidade com a última versão da linguagem Swift, a 4.0;
- Ferramenta de construção de interface gráfica integrada (*Interface Builder*);

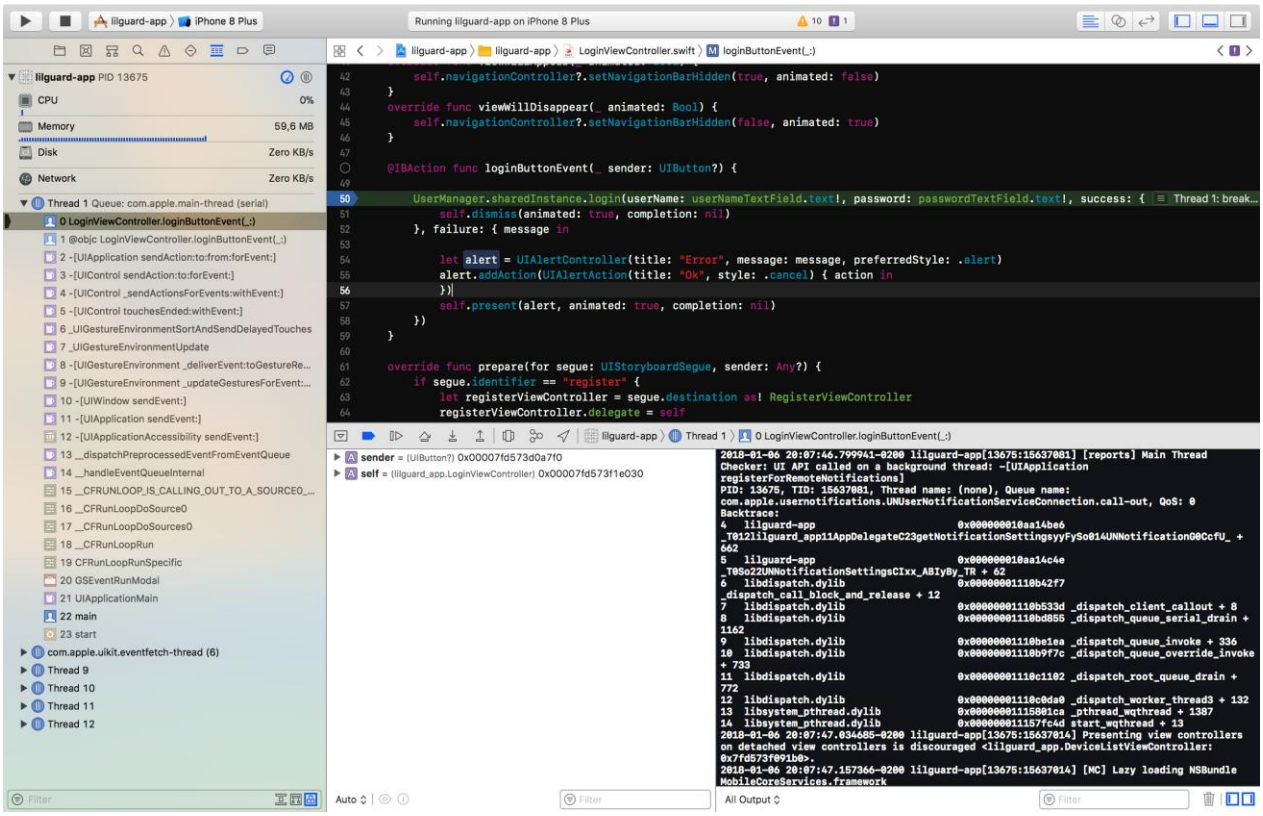
- *Autocomplete* - funcionalidade que completa os nomes de variáveis e métodos automaticamente;
- Documentação de todos os Frameworks do iOS.

Outros pontos fortes do XCode:

- *Breakpoints* programáveis;
- Ferraments de *debug*;
- *Debugging* sem fio via Wi-Fi;
- Visualização da pilha de chamadas;
- Visualização em 3D da estrutura de elementos renderizados nas telas;
- Console de debug LLDB (Low Level Debugger);
- Simuladores de todos os iPhones e iPads pré-instalados.

A Figura 18 mostra a captura de tela da IDE durante uma sessão de debug, com a execução do programa congelada. A pilha de chamadas pode ser visualizada à esquerda, o console de debug está no canto inferior direito e o código na parte superior à direita.

Figura 18: Captura de tela do XCode durante sessão de *debug*.

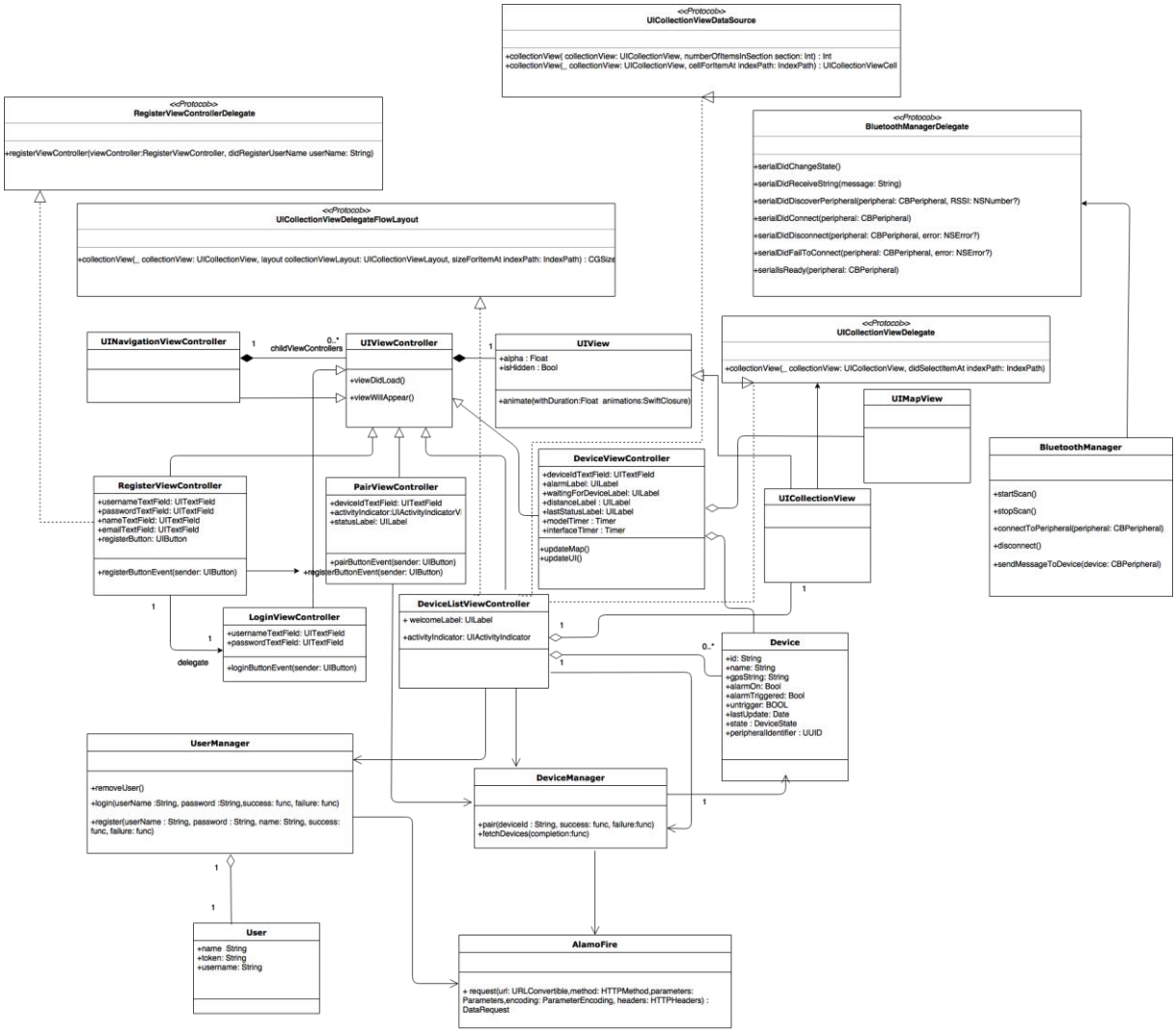


Fonte: Os autores.

3.4.2 Código

O principal padrão de projeto de software da arquitetura dos componentes do sistema é o Model-View-Controller. Todas as classes do *framework* de interface do iOS, o UIKit, aplicam este padrão. Todas as classes referentes à apresentação de elementos nas telas e navegação entre elas seguem este paradigma. Naturalmente, o código desenvolvido neste projeto seguiu o Model-View-Controller para fazer o reuso de componentes padrão de tela do sistema e para garantir a máxima compatibilidade da aplicação com o iOS.

Figura 19: Diagrama de Classes resumido do aplicativo *Pocket Guard*.

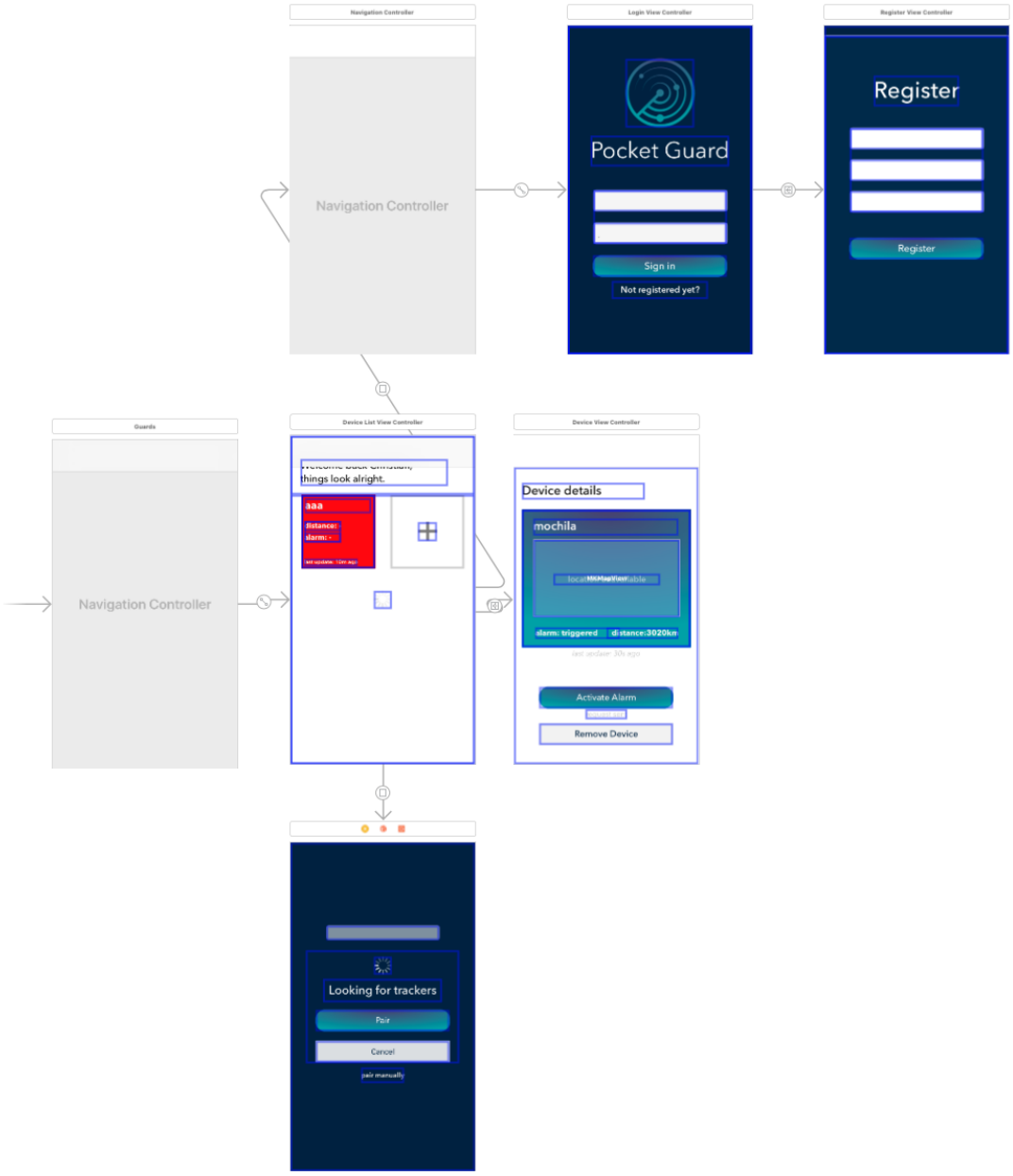


Fonte: Os autores.

A Figura 19 apresenta o diagrama de classes do sistema. As classes e protocolos pertencentes ao UIKit podem ser identificadas pelas iniciais UI. Algumas delas foram suprimidas do diagrama para manter sua simplicidade. Caso contrário, teriam que ser retratados dezenas de níveis de relacionamentos internos do *framework*, irrelevantes para a representação deste projeto. As variáveis e métodos responsáveis por funcionalidades internas de algumas classes, como o método que ajusta a tela para que o teclado não cubra campos de digitação na tela de cadastro de usuário, também foram suprimidas.

As telas da aplicação foram desenvolvidas usando a ferramenta visual *Storyboard*, integrada ao ambiente de desenvolvimento XCode. Na Figura 20 é retratado o *wireframe* de telas do aplicativo.

Figura 20: *Wireframe* de telas do aplicativo.



Fonte: Os autores.

Para realização de tarefas assíncronas, como a atualização periódica da interface de usuário e das informações referentes aos rastreadores, foram usados *timers* do sistema operacional iOS em conjunto com o sistema de despacho de tarefas GCD (*Grand Central Dispatch*), o qual permite que blocos de código sejam alocados em filas para serem executados, evitando condições de concorrência para tarefas que estejam na mesma fila.

3.4.3 Frameworks

No desenvolvimento do aplicativo foram usadas bibliotecas de código, também conhecidas como *frameworks*. O uso deles é obrigatório para acesso a algumas funcionalidades do sistema, como a comunicação Bluetooth e o disparo de notificações ao usuário. Outras bibliotecas foram usadas para agilizar e simplificar o desenvolvimento do software.

O principal *framework* usado na aplicação foi o UIKit. Um dos mais importantes do iOS, fornece todas as classes e interfaces necessárias para a construção da interface de usuário. Botões, formulários, listas, tabelas, imagens, menus, navegação entre telas e muitos outros elementos estão presentes nele. Esta biblioteca foi usada extensivamente por todo o aplicativo.

O CoreData, um *framework* de persistência de grafos de objetos, foi usado para armazenar dados de usuário e persisti-los. Este *framework* é disponibilizado pela Apple no macOS e iOS. As entidades são modeladas pelo desenvolvedor em um editor visual, no qual é possível declarar suas propriedades e as relações entre elas, que incluem associação entre entidades e herança (conceitos da programação orientada a objetos). Durante a execução do código, é possível obter instâncias dessas entidades através de *queries* ou diretamente da associação entre as entidades. Para persistência do grafo de objetos, os dados são serializados e podem ser armazenados de diversos modos, como em um arquivo XML (*Extensible Markup Language*) ou em um banco SQL (*Structured Query Language*), dependendo da configuração realizada.

Para acesso ao sistema de notificações ao usuário na plataforma iOS, o *framework* UserNotifications se faz necessário. Esta biblioteca é essencialmente uma interface para envio de notificações. A Apple centraliza e controla as notificações de todos os aplicativos em um serviço, o *Apple Push Notification Center*, e não permite que sejam enviadas de outra maneira.

A comunicação usando Bluetooth Low Energy foi viabilizada no iOS através do *framework* CoreBluetooth. Ele permite que o dispositivo encontre periféricos Bluetooth, estabeleça uma conexão, descubra seus serviços disponíveis e leia suas características. Ele também dá suporte ao funcionamento em background, desta forma o aplicativo, quando está minimizado, ainda pode receber notificações caso um dispositivo seja encontrado ou desconectado.

Para envio de requisições HTTP ao servidor e tratamento das respectivas respostas, a biblioteca Alamofire foi usada. Escrita em Swift, ela encapsula múltiplas funcionalidades de rede, como verificação do estado da conexão da Internet, montagem de URL (*Uniform Resource Locator*) e corpo de requisições HTTP, codificação e decodificação de mensagens no formato JSON (*Javascript Object Notation*), *upload* e *streaming* de dados. Seu código é *open-source* e está disponível na plataforma GitHub.

Todo o código do aplicativo pode ser encontrado no apêndice A.

3.5 DESENVOLVIMENTO DO SERVIÇO

Para a execução deste projeto, fez-se necessária a criação de um *webserver* para mediar a comunicação entre o dispositivo rastreador e o aplicativo, também para autenticação e armazenamento de dados de usuário, gestão dos dispositivos e disparo de notificações.

A tecnologia usada para desenvolvimento do serviço foi o Node.js, que é um *runtime environment* para execução de Javascript por parte do servidor. Sua arquitetura é direcionada a eventos, o que torna possível a gestão assíncrona de input e output, aumentando assim a capacidade e a escalabilidade do servidor. Uma *thread*

despacha eventos individuais ao retirá-los de uma fila global e os executa um de cada vez, portanto não há necessidade de código para sincronização de acesso a recursos.

A arquitetura que norteou a codificação chama-se *Representational State Transfer*, que nada mais é que uma abstração da arquitetura que rege os *websites* da *World Wide Web*. Os servidores que seguem esta implementação interagem com os clientes da mesma maneira que os servidores que hospedam *websites*, através dos métodos do HTTP, como por exemplo *GET, POST, PUT e DELETE*. Além disso, as respostas das requisições são marcadas pelos códigos HTTP, como por exemplo o 404 para recursos não encontrados, 201 para criação de novos recursos e 500 para erro interno do servidor. Um servidor que segue este padrão é comumente chamado de *RESTful*.

Visando oferecer segurança e privacidade ao usuário, especialmente caso ele esteja usando seu celular numa rede Wi-Fi desprotegida, todas as requisições enviadas pelo *smartphone* ao servidor são criptografadas usando o protocolo HTTPS (*Hypertext Transfer Protocol Secure*). O certificado SSL é gerado pelo serviço de hospedagem usado para o *webserver*, oferecido pela empresa Heroku.

Assim como no desenvolvimento do aplicativo, usaram-se bibliotecas que agilizam e simplificam a criação do código, as quais foram instaladas usando o gestor de pacotes Javascript *npm*. A biblioteca *express*, que executa o roteamento de requisições, foi usada em conjunto com a biblioteca *body-parser* para o tratamento dos dados provenientes delas. Para a autenticação de usuário é usada a biblioteca *json web token*, que gera um *token* com prazo de expiração definido, usando o algoritmo de criptografia HMAC-SHA256. Desta maneira, o usuário não precisa transmitir sua senha sempre que faz um *request*, apenas seu *token*.

A Tabela 4 relaciona as chamadas públicas da API. A Tabela 5 relaciona as chamadas permitidas somente a usuários autenticados.

Tabela 4: API pública do servidor Pocket Guard.

Método HTTP	**Rota**	**Função**
GET	/setup	Faz o *setup* inicial do usuário *admin* (*request* para auxílio no desenvolvimento)
POST	/signup	Faz o cadastro do usuário
POST	/authenticate	Autentica o usuário e retorna seu *token*
GET	/users	Retorna todos os usuários cadastrados (*request* para auxílio no desenvolvimento)
PUT	/eraseUsers	Apaga todos os usuários (*request* para auxílio no desenvolvimento)
POST	/devices	Pré-cadastro dos rastreadores existentes
GET	/deviceStatus	Método que recebe o estado do rastreador como parâmetro e retorna comandos de controle a ele

Fonte: Os autores.

Tabela 5: API privada do servidor Pocker Guard.

Método HTTP	**Rota**	**Função**
POST	/api/pair	Atrelamento de dispositivo a usuário
GET	/api/devices	Lista os dispositivos do usuário
GET	/api/device/:deviceId	Retorna detalhes de um dispositivo específico do usuário
PUT	/api/device/:deviceId	Atualiza nome e cor do dispositivo, manda o evento de *untrigger* para o alarme
POST	/api/unpair	Desatrelamento de dispositivo a usuário

Fonte: Os autores.

O corpo das mensagens é enviado no formato *Javascript Object Notation* (JSON), um formato limpo e humanamente legível de encapsulamento de dados. Ele dá suporte a vetores, dicionários de chave e valor, e tipos puros de dados, como *strings* e números. A Figura 21 apresenta o JSON extraído do *log* de resposta da chamada *GET /api/devices* de um usuário autenticado que possui dois dispositivos.

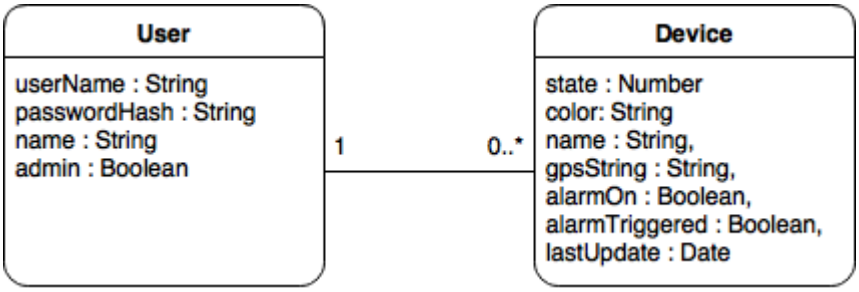
Figura 21: Exemplo de JSON de resposta do servidor.


```
{
   "devices":[
      {
         "_id":"5a5bb5b9c33e89d2cf000004",
         "color":"000fff",
         "alarmOn":false,
         "alarmTriggered":false,
         "state":0,
         "__v":0,
         "name":"device2",
         "user":"5a5bb5a3c33e89d2cf000003"
      },
      {
         "_id":"5a5bb5cbc33e89d2cf000005",
         "color":"000fff",
         "alarmOn":false,
         "alarmTriggered":false,
         "state":0,
         "__v":0,
         "name":"device3",
         "user":"5a5bb5a3c33e89d2cf000003"
      }
   ]
}
```


Fonte: Os autores.

O banco de dados escolhido para armazenamento de dados de usuário e de dispositivos foi o MongoDB (Mongo *DataBase*). Diferentemente dos bancos da família SQL, ele não é baseado em tabelas. Seus dados são armazenados num documento no formato BJSON (*Binary Javascript Object Notation*), portanto todos os tipos de dados do Javascript são nativamente suportados. O armazenamento de usuários e dispositivos no banco de dados foi feito usando a biblioteca *mongoose*, que é essencialmente uma camada de modelamento de objetos sobre o MongoDB. A Figura 22 representa as entidades modeladas para persistência neste projeto.

Figura 22: Entidades modeladas.



Fonte: Os autores.

A hospedagem se deu pela contratação do serviço Heroku, uma plataforma que suporta diversas linguagens de programação, inclusive Javascript. O Heroku foi o escolhido dentre outras opções pela qualidade de seu serviço e pela facilidade para se fazer o *upload* de uma aplicação, assim como pelas ferramentas auxiliares fornecidas gratuitamente, como por exemplo uma interface de linha de comando customizada.

Para o envio de notificações ao usuário, foi criada uma ponte entre o *server Pocket Guard* e o *Apple Push Notification Service*, que é a peça central do sistema de notificações dos dispositivos Apple. Ao fazer seu cadastro na plataforma *Pocket Guard*, o usuário é associado ao *token* de notificações de seu *smartphone*. Uma chave criptografada foi gerada no portal de desenvolvedores da Apple e adicionada ao serviço para autenticação e validação das notificações enviadas.

O código foi desenvolvido no editor de texto TextMate, que dá suporte a indentação e identificação de símbolos do Javascript. O *build* do código se deu via linhas de comando.

Para o controle de versão de código fonte, foi usada o sistema Git, o qual permite manter controle sobre as mudanças ocorridas em arquivos de código e coordenar o trabalho entre múltiplas pessoas, assim como reverter mudanças já feitas. O código do servidor, assim como do aplicativo e do rastreador fizeram uso da tecnologia. O repositório Git remoto foi hospedado gratuitamente na plataforma BitBucket.

Todo o código do servidor pode ser encontrado no apêndice B.

3.6 CONSIDERAÇÕES FINAIS

Este capítulo apresentou um detalhamento da estrutura e das funcionalidades do sistema proposto e implementado chamado Pocket Guard. As etapas de desenvolvimento do protótipo, do aplicativo e do serviço foram detalhadas conforme as necessidades do projeto, culminando em uma única solução, a qual será avaliada e discutida no próximo capítulo.

4 RESULTADOS

4.1 CONSIDERAÇÕES INICIAIS

Como apresentado anteriormente, o Pocket Guard pode ser usado para o monitoramento de pertences pessoais ou objetos aos quais o protótipo possa ser afixado. Para melhor demonstrar os resultados obtidos, este capítulo vai apresentar cenários reais de aplicação do dispositivo acompanhados de uma discussão das principais características observadas. Além disso, apresentará uma avaliação da usabilidade do protótipo e do aplicativo, e também uma avaliação sobre o consumo de energia.

4.2 USABILIDADE

A versão final deste projeto não apresentou problemas no tocante à usabilidade. O aplicativo está responsivo, não apresentando atrasos ou travamentos na troca de telas, graças ao gerenciamento eficiente de *threads*. A interface, enxuta, permite ao usuário realizar as atividades de cadastro, pareamento e monitoramento.

A notificação de saída de proximidade foi recebida sempre que o rastreador foi retirado dos arredores do usuário, procedimento testado recorrentemente. A notificação de movimentação do objeto, quando este está distante do usuário, também foi entregue com sucesso, sempre com atraso maior, pois requer a ativação do módulo GPRS e, consequentemente, a busca pelo sinal da torre mais próxima.

O algoritmo de detecção de movimentação teve resultados corretos, sensíveis o suficiente para detectar movimentações estranhas e gerar a notificação de disparo, mas não para gerar falsos positivos em função do ruído quando o rastreador está parado. No entanto, existe uma limitação: sempre que o dispositivo está enviando a mensagem de atualização ao servidor, processo que leva alguns segundos, não são feitas leituras do acelerômetro. Uma movimentação rápida que movesse o rastreador para outra posição e mantivesse o seu ângulo em relação ao solo, não geraria disparo.

Como o servidor se trata de uma versão grátis, ele é colocado em modo *sleep* após 30 minutos de atividade. A consequência disso é que a primeira requisição a ser recebida terá um tempo de resposta prolongado, perceptível ao usuário final ao abrir o aplicativo. Sendo assim, considerando que esta é uma prova de conceitos, este tempo de atraso inicial não é relevante.

Devido ao gerenciamento agressivo de bateria, que desativa o rádio GPRS quando o dispositivo está ativo e agenda a comunicação para um intervalo de 5 minutos, alguns comandos podem chegar com atraso ao dispositivo, como por exemplo a notificação de *unpair*, que ocorre quando um usuário remove um rastreador da sua lista de dispositivos.

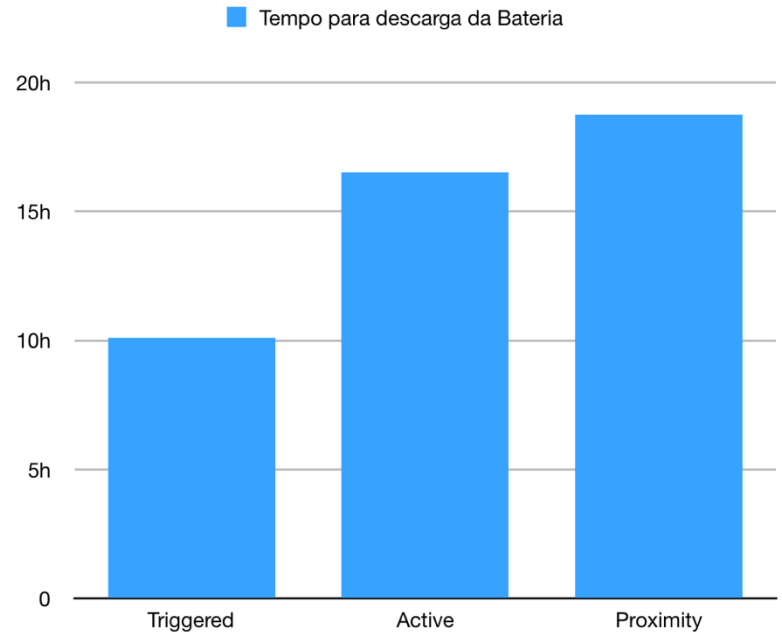
4.3 DURAÇÃO DA BATERIA

Uma longa duração de bateria do dispositivo é um dos objetivos deste projeto, para que o usuário não precise recarregá-lo ao longo do dia. Para isso, foram criados dois estados de baixo consumo de energia. O primeiro deles é o estado de proximidade, que detecta a presença do *smartphone* do usuário através da tecnologia Bluetooth, desativa qualquer outra comunicação via dados móveis (GPRS) e o sistema de posicionamento global GLONASS, visando o máximo de economia de energia. E o segundo deles é o estado ativo, que num primeiro momento ativa o módulo Sim808, responsável pela comunicação GPRS e GNSS, o desativa e volta a ativá-lo a cada 5 minutos, visando economizar energia.

Para apurar os resultados de consumo de energia em cada um dos estados do dispositivo, foram realizados testes usando uma bateria externa. A bateria usada, da marca BRIGHT, com capacidade de carga nominal de 2600 mAh a 5 V, resultando em 13 Wh, peso de aproximadamente 100g e dimensões de 96 x 24 x 22 mm, seria uma opção adequada caso este protótipo venha a se tornar um produto comercial devido ao seu peso e tamanho reduzidos, que permitiriam a miniaturização do produto final. No entanto, é de praxe que os fabricantes de baterias inflem as especificações de carga com o propósito de aumentar as vendas, também é comum que a bateria perca sua

capacidade naturalmente com o tempo, por isso, neste trabalho, a capacidade é tratada como menor ou igual à especificação do fabricante. Como referência, a bateria do celular iPhone 8, com dimensões similares, possui apenas 6,96 Wh de carga. Foram feitos testes de descarregamento com o dispositivo em seus três estados. A Figura 23 apresenta o tempo de descarga da bateria em cada um deles.

Figura 23: Comparativo de descarga de bateria entre estados.



Fonte: Os autores.

O consumo de energia dos estados *triggered*, *active* e *proximity* (respectivamente, estados de disparo, ativo e de proximidade) foi calculado a partir destes dados e resultou em ≤ 1,29 W, ≤ 0,78 W e ≤ 0,69 W, para cada estado, respectivamente. Fica evidente o ganho de duração de bateria do dispositivo devido aos seus estados de economia de energia *active* e *proximity*.

Conclui-se que o protótipo apresenta consumo de energia adequado. A duração da carga é suficiente para dar horas de autonomia ao rastreador com baterias comuns, devido aos baixos níveis de consumo nos estados *active* e *proximity*, que

provavelmente seriam os mais usados em um cenário real. O dispositivo pode então passar um dia monitorando o objeto de interesse, caso seja ativado pela manhã, e terá autonomia até a noite, já que seus estados de baixo consumo duram de 16 a 18 horas corridas.

4.4 ESTUDOS DE CASO

Para validação dos resultados e testes finais, o *Pocket Guard* foi usado em situações do cotidiano para as quais foi concebido, resultando em três estudos de caso.

O primeiro estudo de caso foi o de monitoramento de bolsas e malas, no qual o sistema se mostrou extremamente útil. O rastreador foi posicionado dentro de uma bolsa e permaneceu nela ao ser carregada durante o dia, para simular uma situação cotidiana de muitos estudantes e trabalhadores pelo mundo. A Figura 24 apresenta o dispositivo posicionado dentro de uma mochila.

Figura 24: Rastreador posicionado em uma mochila.

Fonte: Os autores.

A presença do rastreador provê ao usuário segurança quanto à proximidade da mochila. Em caso de furto ou esquecimento, o usuário é avisado quando a mochila deixa sua proximidade, após poucos segundos. Caso o usuário deseje deixar a mochila em algum local, consegue fazê-lo com mais segurança, já que será notificado se alguém movimentar o seu pertence. A limitação é que o *smartphone* deve estar sempre sendo portado pelo usuário e não pode ser deixado dentro da mochila, o que não é crítico, já que o usuário precisaria portar o celular de qualquer maneira para receber as notificações.

Em uma simulação de furto, o dispositivo respondeu como esperado, enviando uma notificação de movimentação ao *smartphone* do usuário alguns segundos após a mochila ser movimentada fora de sua proximidade. O protótipo conseguiu estabelecer comunicação GPRS para envio de notificações mesmo com a bolsa fechada. O GNSS, no entanto, não conseguiu definir a localização do rastreador. Tal problema poderia ser resolvido em uma futura miniaturização, de maneira que o rastreador pudesse ser

acoplado à mochila como um chaveiro, tendo assim espaço livre acima de si para funcionamento do GNSS.

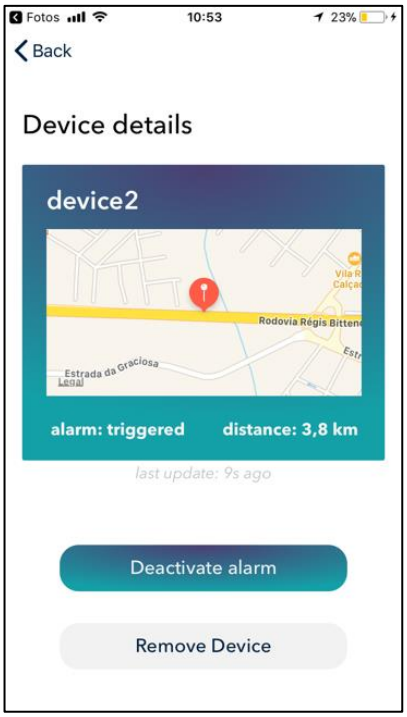
Um segundo estudo de caso foi efetuado com o monitoramento de um automóvel, o qual também apresentou resultados satisfatórios. Ao posicionar o rastreador Pocket Guard no interior do carro, o usuário pode controlar sua posição e movimentação.

Figura 25: Rastreador posicionado em um automóvel.

Fonte: Os autores.

Em testes efetuados, a aceleração do carro foi detectada corretamente e o disparo foi efetuado quando o *smartphone* do usuário não se encontrava por perto. O Pocket Guard possibilita assim maior controle de um automóvel entregue a terceiros, como por exemplo manobristas de restaurantes, assim como no caso de roubo do automóvel. A Figura 26 mostra uma captura de tela do aplicativo mostrando a situação de um rastreador posicionado dentro de um automóvel, em seu estado de disparo.

Figura 26: Captura de tela do aplicativo mostrando disparo do rastreador e a localização do pertence.



Fonte: Os autores.

Para que o GNSS determine a localização do dispositivo, ele tem que ser posicionado em algum ponto do carro próximo às janelas e que não seja coberto, evitando locais como embaixo de bancos ou no porta-luvas.

Um estudo de caso adicional foi efetuado no monitoramento de acesso a cômodos de uma residência, via acoplamento do rastreador a portas, o qual também apresentou resultados satisfatórios. O usuário pode ser informado em segundos sobre a abertura de portas em sua residência. No entanto, o Pocket Guard não pode ser considerado um sistema completo de monitoramento residencial, já que não conta com sensores de presença, por exemplo.

4.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou uma avaliação dos resultados obtidos com o Pocket Guard. Em termos de usabilidade, algumas limitações foram apontadas, mas não invalidam a prova de conceito que era objetivo do protótipo implementado. A estratégia para redução do consumo de energia na implementação do protótipo foi validada com experimentos. E, por fim, alguns estudos de caso com potenciais aplicações do protótipo foram apresentados e discutidos.

5 CONCLUSÕES

O objetivo deste trabalho constituía-se no desenvolvimento de um protótipo de um sistema de monitoramento IoT, o qual foi alcançado satisfatoriamente. Usando conceitos de arquitetura IoT, descobertos após pesquisa por artigos publicados sobre o estado da arte na área, foi criada uma solução completa de rastreamento portátil, que se baseia em um aplicativo e um rastreador, conectados por um serviço web. Como diferenciais das soluções de monitoramento e rastreamento no mercado, destacam-se a portabilidade do sistema, o aplicativo e o baixo consumo de energia.

A arquitetura criada centraliza a lógica de aplicação no serviço, assim como a comunicação entre *smartphone* e rastreador, através de uma API RESTful. Contudo, em alguns momentos oportunos, também ocorre comunicação direta entre ambos via Bluetooth Low Energy.

Através de estados de baixo consumo de energia foi possível reduzir o consumo de energia do protótipo de rastreador, ao custo de latência na comunicação com o servidor, permitindo alguns dias de duração com uma bateria de pequena carga. Para obter esse resultado foram usadas algumas soluções inteligentes, como detecção de presença do usuário dono do rastreador, desativando funcionalidades e reduzindo o consumo de energia ao mínimo possível quando ele estiver próximo.

Para a execução deste trabalho foram utilizados conceitos aprendidos ao longo da graduação em Engenharia Eletrônica, na área de sistemas embarcados, microcontroladores, fundamentos de programação, comunicações digitais, desenvolvimento de software, Internet das coisas, entre outros. Todos estes conceitos são importantes para fundamentar o conhecimento de um Engenheiro Eletrônico preparado para as novas tecnologias que estão sendo desenvolvidas na área.

Ainda há muito o que melhorar para o desenvolvimento de um produto final, como será explicado na seção de Trabalhos Futuros, porém, os resultados alcançados

já se sustentam como prova de conceito, podendo ser utilizado como fundamento em um futuro projeto.

Por fim, este trabalho permitiu, também, adquirir conhecimentos teóricos e práticos acerca do desenvolvimento de sistemas sob a ótica da Internet das coisas, área que está em franco desenvolvimento.

5.1 DIFICULDADES ENCONTRADAS

Naturalmente, algumas dificuldades foram encontradas no desenvolvimento deste projeto, sendo a maioria delas durante a codificação do rastreador. Para o seu hardware, fez-se a escolha pelo Arduino Uno, devido ao seu baixo custo e outras razões previamente citadas, apesar do poder de processamento limitado e memória RAM ínfima. Seu núcleo é um processador ARM com *clock* de 16 MHz e a memória RAM é de apenas 2 KB. Por causa destas limitações, durante o desenvolvimento do código, algumas variáveis não estavam mais sendo alocadas, pois a quantidade máxima de memória já estava sendo usada. O código teve que ser reescrito e otimizado em alguns pontos para reduzir o uso de RAM, como por exemplo no momento em que é enviada a requisição de atualização para o servidor. Neste momento, a *string* passava por diversas concatenações, operação taxativa em uso de memória. A solução encontrada foi transmitir os parâmetros um a um diretamente via comunicação serial, deixando partes da *string hard coded* no código do programa.

Outra dificuldade encontrada foi na implementação do sistema de posicionamento. O *shield* Sim808 dá suporte tanto ao sistema GPS, pertencente ao governo americano, quanto ao GLONASS, pertencente ao russo. Por alguma razão desconhecida, o módulo retornava coordenadas errôneas ao se usar o GPS, com mais de 15 km de erro. Ao migrar-se para o sistema GLONASS, o sistema apresentou precisão adequada, com erros próximos a 10 m em espaço aberto.

No desenvolvimento do aplicativo, o desafio foi gerenciar a assincronicidade das respostas das requisições enviadas ao servidor, que poderiam gerar condições de concorrência no armazenamento e leitura dos dados armazenados. Para resolver este

problema, foi usada a tecnologia GCD de despacho de tarefas, que evita tais condições.

5.2 TRABALHOS FUTUROS

Existem diversas maneiras de melhorar o protótipo e o sistema como um todo futuramente. Uma delas, e de relativa maior importância caso haja a intenção de transformar este projeto em um produto real no mercado, é a “apresentação” do dispositivo. Deve-se desenvolver uma proteção de plástico para abrigar o circuito e miniaturizar o protótipo, idealmente para que ele alcance o tamanho de um chaveiro, de modo a aumentar sua discrição e, consequentemente, suas aplicações. Para isso é importante, além de diminuir o tamanho do circuito em si, utilizar uma bateria pequena.

Outra questão importante a se melhorar é a comunicação entre o dispositivo, o *smartphone* e o servidor, procurando reduzir os atrasos e criptografar as mensagens, de modo a aumentar a segurança e a confiabilidade do sistema.

Seria, também, interessante a implementação do *firmware* do dispositivo utilizando um RTOS (*Real-Time Operating System*), dividindo em diversas tarefas com diferentes graus de prioridade, fazendo com que o sistema trabalhe de maneira mais eficiente.

Mais uma melhoria seria adicionar um controle de carga restante existente na bateria, usando para isso o conversor analógico-digital da placa, para que o usuário possa estar ciente quanto à expectativa de duração de carga.

Por último, visando alcançar o maior número de usuários possível, seria importante desenvolver o aplicativo também para *smartphones* Android e Windows Phone.

REFERÊNCIAS BIBLIOGRÁFICAS

ABEDIN, Sarder F.; ALAM, Golam R.; HONH, Choong S. **A system model for energy efficient green-IoT network**. International Conference on Information Networking, 2015, Camboja.

APPLE INC. **About Objective-C**. 2014. Disnponível em: <https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>. Acesso em: 15 dez. 2017.

_____. **The swift programming language**. Swift 4.0.3 ed. [S.l.]: Apple Inc., 2017.

APPEGO. **Appego GPS:** Rastreamento e monitoramento multifuncional. Disponível em: <http://www.meappego.com.br>. Acesso em: 2 fev. 2018.

ARDUINOJSON. **C++ JSON library for IoT. Simple and efficient**. Disponível em: <https://github.com/bblanchon/ArduinoJson>. Acesso em: 15 dez. 2017.

AUTOTRAC. **Monitoramento e rastreamento de frotas**. Disponível em: <http://www.autotrac.com.br>. Acesso em: 2 fev. 2018.

BLUETOOTH. **Legacy core specifications**. Disponível em: <https://www.bluetooth.com/specifications/bluetooth-core-specification/legacy-specifications>. Acesso em: 10 fev. 2018

_____. **From prototype to global standard – Bluetooth celebrates 20 year of Market creation**, 2018. Disponível em: <https://www.bluetooth.com/news/pressreleases/2018/01/from-prototype-to-global-standard-bluetooth-celebrates-20-years-of-market-creation>. Acesso em: 10 fev. 2018.

CAI, Jian; GOODMAN, David J. **General Packet Radio Service in GSM**. IEEE Communications Magazine, IEEE, v. 35, n. 10, p. 122–131, Outubro de 1997.

CHAN, Steven; CONNELL, Adam; MADRID, Eribel; PARK, Dongkuk; KAMOUA, Ridha. **RFID for Personal Asset Tracking**. IEEE Applications and Technology Conference, 2009, Long Island, Estados Unidos.

EETIMES. **The why, where and what of low-power SoC design**, 2004. Disponível em: <https://www.eetimes.com/document.asp?doc_id=1276973>. Acesso em: 24 jan. 2018

FOLHA DE SÃO PAULO. **Brasil tem um roubo ou furto de veículo a cada minuto; Rio lidera o ranking**, 2017. Disponível em: <http://www1.folha.uol.com.br/cotidiano/2017/10/1931061-brasil-tem-1-roubo-ou-furto-de-veiculo-a-cada-minuto-rio-lidera-o-ranking.shtml>. Acesso em: 24 jan. 2018.

GARTNER. **Gartner says 8,4 billion connected "things" will be in 2017, up 31 percent from 2016**, 2017. Disponível em: <https://www.gartner.com/newsroom/id/3598917>. Acesso em 10 de fevereiro de 2018.

GOUY, Isaac. **The computer language benchmarks game**, 2017. Disponível em: <https://benchmarksgame.alioth.debian.org/u64q/measurements.php?lang=swift>. Acesso em: 15 dez. 2017.

HRIBERNIK, Karl A. et al. **Co-creating the Internet of things:** First experiences in the participatory design of intelligent products with Arduino. IEEE Conference, 2011, Aachen, Alemanha.

KOVATSCH, Matthias; MAYER, Simon; OSTERMAIER, Benedikt. **Moving application logic from the firmware to the cloud:** towards the thin server architecture for the Internet of things. IEEE Conference, 2012, Palermo, Itália.

KORHONEN, Jouni et al. **Measure performance of GSM, HSCSD and GPRS**. In: IEEE Conference, 2001, Helsink, Finlândia.

IOT Analytics. **The 10 most popular Internet of things applications right now**, 2016. Disponível em: <https://iot-analytics.com/10-Internet-of-things-applications/>. Acesso em> 02 fev. 2018.

PIYARE, Rajeev. **Internet of things: ubiquitous home control and monitoring system using android based smart phone.** International Journal of Internet of Things, v. 2, n. 1, p. 5-11, 2013.

POPE, Stephen. **A cookbook for using the model – view controller user interface paradigm in smalltalk – 80**, 1998. Disponível em: <https://www.researchgate.net/publication/248825145_A_cookbook_for_using_the_model_-_view_controller_user_interface_paradigm_in_Smalltalk_-_80>. Acesso em: 24 jan. 2018.

ROTH, Graham. **Bluetooth wireless technology**, 2013. Disponível em: <http://large.stanford.edu/courses/2012/ph250/roth1/>. Acesso em: 10 fev. 2018.

TIM. **Mapa de cobertura**, 2017. Dísponível em: <http://www.tim.com.br/pr/para-voce/cobertura-e-roaming/mapa-de-cobertura>. Acesso em 11 de novembro de 2017.

TILKOV, Stefan; VINOSKI, Steve. **Node.js: Using JavaScript to build high-performance network programs**. IEEE Internet Computing, v. 14, n. 6, p. 80 – 83, 2010.

TRACKR. **Works on keys, wallets & more**. Disponível em: < https://secure.thetrackr.com/>. Acesso em: 2 fev. 2018.

UFAL. **Pesquisador mostra como a Internet das coisas é útil para resolver problemas do cotidiano**, 2016. Disponível em: < http://www.ufal.edu.br/noticias/2016/12/pesquisador-mostra-como-a-Internet-das-coisas-e-util-para-resolver-problemas-do-cotidiano>. Acesso em: 02 fev. 2018.

VATHARE, Rushikesh S. et al. **Comparative study of messaging protocols for the Internet of things**. International Engineering Research Journal, Special Issue, p. 467-471, 2017.

ZHENG, Weifeng; WANG, Xuan; KAMOUA, Ridha. **Personal asset tracking**. IEEE Applications and Technology Conference, 2013, Long Island, Estados Unidos.

APÊNDICE A – CÓDIGO FONTE DO APLICATIVO

```
//
//  BluetoothManager.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 19/11/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
//

import UIKit
import CoreBluetooth



protocol BluetoothManagerDelegate {

    func serialDidChangeState()
    func serialDidDisconnect(_ peripheral: CBPeripheral, error: NSError?)
    func serialDidReceiveString(_ message: String)
    func serialDidDiscoverPeripheral(_ peripheral: CBPeripheral, RSSI: NSNumber?)
    func serialDidConnect(_ peripheral: CBPeripheral)
    func serialDidFailToConnect(_ peripheral: CBPeripheral, error: NSError?)
    func serialIsReady(_ peripheral: CBPeripheral)
}


extension BluetoothManagerDelegate {
    func serialDidReceiveString(_ message: String) {}
    func serialDidDiscoverPeripheral(_ peripheral: CBPeripheral, RSSI: NSNumber?) {}
    func serialDidConnect(_ peripheral: CBPeripheral) {}
    func serialDidFailToConnect(_ peripheral: CBPeripheral, error: NSError?) {}
    func serialIsReady(_ peripheral: CBPeripheral) {}
}


final class BluetoothManager: NSObject, CBCentralManagerDelegate, CBPeripheralDelegate {



    static let sharedInstance = BluetoothManager()

    var delegate: BluetoothManagerDelegate?


    var centralManager: CBCentralManager!


    var pendingPeripheral: CBPeripheral?


    var connectedPeripheral: CBPeripheral?


    weak var writeCharacteristic: CBCharacteristic?


    var isReady: Bool {
        get {
            return centralManager.state == .poweredOn &&
                connectedPeripheral != nil &&
```

```
            writeCharacteristic != nil
        }
    }

    var isScanning: Bool {
        return centralManager.isScanning
    }

    var isPoweredOn: Bool {
        return centralManager.state == .poweredOn
    }

    var serviceUUID = CBUUID(string: "FFE0")

    var characteristicUUID = CBUUID(string: "FFE1")

    private var writeType: CBCharacteristicWriteType = .withoutResponse

    override init() {
        super.init()
        centralManager = CBCentralManager(delegate: self, queue: nil)
    }

    func startScan() {
        guard centralManager.state == .poweredOn else { return }
        centralManager.scanForPeripherals(withServices: [serviceUUID], options: nil)

        let peripherals = centralManager.retrieveConnectedPeripherals(withServices: [serviceUUID])
        for peripheral in peripherals {
            delegate?.serialDidDiscoverPeripheral(peripheral, RSSI: nil)
        }
    }

    func stopScan() {
        centralManager.stopScan()
    }

    func connectToPeripheral(_ peripheral: CBPeripheral) {
        pendingPeripheral = peripheral
        centralManager.connect(peripheral, options: nil)
    }

    func disconnect() {
        if let p = connectedPeripheral {
            centralManager.cancelPeripheralConnection(p)
        } else if let p = pendingPeripheral {
            centralManager.cancelPeripheralConnection(p)
        }
    }

    func sendMessageToDevice(_ message: String) {

        guard isReady else { return }
```

```
        if let data = message.data(using: String.Encoding.utf8) {
            connectedPeripheral!.writeValue(data, for: writeCharacteristic!, type: writeType)
        }
    }


    func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi
RSSI: NSNumber) {

        delegate?.serialDidDiscoverPeripheral(peripheral, RSSI: RSSI)
    }

    func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {

        peripheral.delegate = self
        pendingPeripheral = nil
        connectedPeripheral = peripheral

        delegate?.serialDidConnect(peripheral)

        peripheral.discoverServices([serviceUUID])
    }

    func centralManager(_ central: CBCentralManager, didDisconnectPeripheral peripheral: CBPeripheral, error: Error?) {
        connectedPeripheral = nil
        pendingPeripheral = nil


        delegate?.serialDidDisconnect(peripheral, error: error as NSError?)
    }

    func centralManager(_ central: CBCentralManager, didFailToConnect peripheral: CBPeripheral, error: Error?) {
        pendingPeripheral = nil


        delegate?.serialDidFailToConnect(peripheral, error: error as NSError?)
    }

    func centralManagerDidUpdateState(_ central: CBCentralManager) {

        connectedPeripheral = nil
        pendingPeripheral = nil
        delegate?.serialDidChangeState()
    }



    func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {

        for service in peripheral.services! {
            peripheral.discoverCharacteristics([characteristicUUID], for: service)
        }
    }

    func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService, error: Error?) {

        for characteristic in service.characteristics! {
            if characteristic.uuid == characteristicUUID {

                peripheral.setNotifyValue(true, for: characteristic)
```

```
                writeCharacteristic = characteristic
                writeType = characteristic.properties.contains(.write) ? .withResponse : .withoutResponse
                delegate?.serialIsReady(peripheral)
            }
        }
    }

    func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?) {


        let data = characteristic.value
        guard data != nil else { return }

        if let str = String(data: data!, encoding: String.Encoding.utf8) {
            delegate?.serialDidReceiveString(str)
        } else {

        }

        var bytes = [UInt8](repeating: 0, count: data!.count / MemoryLayout<UInt8>.size)
        (data! as NSData).getBytes(&bytes, length: data!.count)
    }

}

//
//  DateOffset.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 30/11/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
//

import Foundation

extension Date {

    func offsetFrom(date: Date) -> String {

        let dayHourMinuteSecond: Set<Calendar.Component> = [.day, .hour, .minute, .second]
        let difference = NSCalendar.current.dateComponents(dayHourMinuteSecond, from: date, to: self);

        let seconds = "\(difference.second ?? 0)s"
        let minutes = "\(difference.minute ?? 0)m" + " " + seconds
        let hours = "\(difference.hour ?? 0)h" + " " + minutes
        let days = "\(difference.day ?? 0)d" + " " + hours

        if let day = difference.day, day          > 0 { return days }
        if let hour = difference.hour, hour       > 0 { return hours }
        if let minute = difference.minute, minute > 0 { return minutes }
        if let second = difference.second, second > 0 { return seconds }
        return ""
    }

}


//
//  DateOffset.swift
//  lilguard-app
```

```
//
//  Created by Christian Becker Pepino on 30/11/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
//

import Foundation

extension Date {

    func offsetFrom(date: Date) -> String {

        let dayHourMinuteSecond: Set<Calendar.Component> = [.day, .hour, .minute, .second]
        let difference = NSCalendar.current.dateComponents(dayHourMinuteSecond, from: date, to: self);

        let seconds = "\(difference.second ?? 0)s"
        let minutes = "\(difference.minute ?? 0)m" + " " + seconds
        let hours = "\(difference.hour ?? 0)h" + " " + minutes
        let days = "\(difference.day ?? 0)d" + " " + hours

        if let day = difference.day, day          > 0 { return days }
        if let hour = difference.hour, hour       > 0 { return hours }
        if let minute = difference.minute, minute > 0 { return minutes }
        if let second = difference.second, second > 0 { return seconds }
        return ""
    }

}


//
//  GuardServiceRoutes.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 12/10/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
//

import Foundation

class GuardServiceRoutes  {
    //static let host = "http://localhost:8080"
    static let host = "http://guard-app.herokuapp.com"

    static let signUpRoute = host + "/signup"
    static let authenticateRoute = host + "/authenticate"
    static let devicesRoute = host + "/api/devices"
    static let deviceRoute = host + "/api/device"
    static let pairRoute = host + "/api/pair"
    static let unpairRoute = host + "/api/unpair"
    static let notificationToken = host + "/api/notificationToken"

}


//
//  UserManager.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 12/10/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
```

```
//
import UIKit
import CoreData
import Alamofire
class UserManager {

    static let sharedInstance = UserManager()

    var user : User? {
        get {
          let delegate = UIApplication.shared.delegate as! AppDelegate
          let moc = delegate.persistentContainer.viewContext
            let request : NSFetchRequest<User> = User.fetchRequest()

            do{
                let users = try moc.fetch(request)
                return users.first
            } catch {
                return nil
            }
        }
    }

    func removeUser() {
        let delegate = UIApplication.shared.delegate as! AppDelegate
        let moc = delegate.persistentContainer.viewContext
        let request : NSFetchRequest<User> = User.fetchRequest()
        do{
            let users = try moc.fetch(request)
            for user in users {
                moc.delete(user)
            }
            try moc.save()
        } catch {
        }
    }

    func login(userName : String, password : String,  success: @escaping ()->(), failure: @escaping (String?)->()) {

        let parameters: Parameters = [
            "userName": userName,
            "password" :  password
        ]

        Alamofire.request(GuardServiceRoutes.authenticateRoute, method: .post, parameters: parameters).responseJSON { response
in
            print("Request: \(String(describing: response.request))")   // original url request
            print("Response: \(String(describing: response.response))") // http url response
            print("Result: \(response.result)")                         // response serialization result

            if let json = response.result.value {
                let jsonDic = json as! [String : Any]
                if jsonDic["success"] as! Bool == true {
                    self.removeUser()
                    let entity =
                        NSEntityDescription.entity(forEntityName: "User",
                                                   in: AppDelegate.moc)!

                    let user = NSManagedObject(entity: entity,
                                               insertInto: AppDelegate.moc) as! User
```

```
            let userDic : [String : Any] = jsonDic["user"] as! [String : Any]
            user.name = userDic["name"] as? String
            user.userName = userDic["userName"] as? String
            user.token = jsonDic["token"] as? String



            try? AppDelegate.moc.save()

            success()
            } else {
                failure(jsonDic["message"] as? String)
            }


        } else {
            failure((response.error?.localizedDescription))
        }

        if let data = response.data, let utf8Text = String(data: data, encoding: .utf8) {
            print("Data: \(utf8Text)") // original server data as UTF8 string
        }
    }

}

func register(userName : String, password : String, name: String, success: @escaping (String)->(), failure: @escaping (String?)-
>()) {

    let parameters: Parameters = [
        "userName": userName,
        "password" :  password,
        "name" : name
    ]

    Alamofire.request(GuardServiceRoutes.signUpRoute, method: .post, parameters: parameters).responseJSON { response in
        print("Request: \(String(describing: response.request))")   // original url request
        print("Response: \(String(describing: response.response))") // http url response
        print("Result: \(response.result)")                         // response serialization result

        if let json = response.result.value {
            let jsonDic = json as! [String : Any]
            if jsonDic["success"] as! Bool == true {

                let userName = jsonDic["userName"] as? String
                success(userName!)
            } else {
                failure(jsonDic["message"] as? String)
            }


        } else {
            failure((response.error?.localizedDescription))
        }

        if let data = response.data, let utf8Text = String(data: data, encoding: .utf8) {
            print("Data: \(utf8Text)") // original server data as UTF8 string
        }
    }

}
```

```
    func sendNotificationToken(token:String) {
        let parameters : Parameters = [
            "notificationToken" : token,
            "token" : user?.token as! String
        ]
        if self.user != nil {
            Alamofire.request(GuardServiceRoutes.notificationToken, method: .post, parameters:
parameters).responseJSON(completionHandler: { response in

                if let json = response.result.value {
                    let jsonDic = json as! [String : Any]
                    if jsonDic["success"] as! Bool == true {
                        print("token notification sent")
                    } else {
                        print("token notification could not be sent")
                    }
                } else {
                        print("token notification could not be sent")
                }

            });
        }
    }
}


//
//  DeviceManager.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 19/11/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
//

import Alamofire

enum DeviceFetchError : Error {
    case expiredToken
    case unknown
}

class DeviceManager {

    static let sharedInstace = DeviceManager()

    func pair(deviceId : String, success: @escaping ()->(), failure: @escaping (String?)->()) {

        let userToken = UserManager.sharedInstance.user?.token
        let parameters: Parameters = [
            "deviceId": deviceId,
            "token" : userToken as! String
        ]

        Alamofire.request(GuardServiceRoutes.pairRoute, method: .put, parameters: parameters).responseJSON { response in

            print("Request: \(String(describing: response.request))")   // original url request
            print("Response: \(String(describing: response.response))") // http url response
            print("Result: \(response.result)")                         // response serialization result

            if let json = response.result.value {
```

```
            let jsonDic = json as! [String : Any]

            if jsonDic["success"] as! Bool == true {
                success()
            } else {
                failure(jsonDic["message"] as! String)
            }
        }
    }
}

func unpair(deviceId : String, success: @escaping ()->(), failure: @escaping (String?)->()) {

    let userToken = UserManager.sharedInstance.user?.token
    let parameters: Parameters = [
        "deviceId": deviceId,
        "token" : userToken!
    ]

    Alamofire.request(GuardServiceRoutes.unpairRoute, method: .put, parameters: parameters).responseJSON { response in

        print("Request: \(String(describing: response.request))")   // original url request
        print("Response: \(String(describing: response.response))") // http url response
        print("Result: \(response.result)")                         // response serialization result

        if let json = response.result.value {
            let jsonDic = json as! [String : Any]

            if jsonDic["success"] as! Bool == true {
                success()
            } else {
                failure(jsonDic["message"] as! String)
            }
        }
    }
}



func fetchDevices(completion:@escaping ([Device]?, DeviceFetchError?)->()) {

    let token = UserManager.sharedInstance.user?.token!
    Alamofire.request(GuardServiceRoutes.devicesRoute + "?token=\(token!)").responseJSON { response in
        print("Request: \(String(describing: response.request))")   // original url request
        print("Response: \(String(describing: response.response))") // http url response
        print("Result: \(response.result)")                         // response serialization result

        if let json = response.result.value {

            let jsonDic = json as! [String : Any]

            if(response.response?.statusCode == 401) {
                completion(nil,DeviceFetchError.expiredToken)
                return
            }

            let devicesDic = jsonDic["devices"] as! [[String : Any]]
            var devices = [Device]()
            for deviceDic in devicesDic {
                let id  = deviceDic["_id"] as! String
                let name = deviceDic["name"] as? String
```

```
                let device = Device(id: id, name: name)
                device.gpsString = deviceDic["gpsString"] as? String
                device.alarmOn = (deviceDic["alarmOn"] as? Bool)!
                device.alarmShouldBeOn = (deviceDic["alarmShouldBeOn"] as? Bool)!
                device.untrigger = (deviceDic["untrigger"] as? Bool)!
                device.state = (deviceDic["state"] as? Int).map {DeviceState(rawValue: $0) }!

                if let lastUpdateString = deviceDic["lastUpdate"] {
                    let dateFormatter = DateFormatter()
                    dateFormatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSSZ"
                    device.lastUpdate = dateFormatter.date(from: lastUpdateString as! String)
                }



                devices.append(device)

            }
            completion(devices,nil)
        } else {
            completion(nil,DeviceFetchError.unknown)
        }



        if let data = response.data, let utf8Text = String(data: data, encoding: .utf8) {
            print("Data: \(utf8Text)") // original server data as UTF8 string
        }
    }
}

func fetchDeviceWithId(deviceId : String, completion:@escaping (Device?, DeviceFetchError?)->()) {

    let token = UserManager.sharedInstance.user?.token!
    Alamofire.request(GuardServiceRoutes.deviceRoute + "/" + deviceId + "?token=\(token!)").responseJSON { response in
        print("Request: \(String(describing: response.request))")   // original url request
        print("Response: \(String(describing: response.response))") // http url response
        print("Result: \(response.result)")                         // response serialization result

        if let json = response.result.value {

            var jsonDic = json as! [String : Any]

            if(response.response?.statusCode == 401) {
                completion(nil,DeviceFetchError.expiredToken)
                return
            }
                jsonDic = jsonDic["device"] as! [String : Any]
                let _id  = jsonDic["_id"] as! String
                let name = jsonDic["name"] as? String
                let device = Device(id: _id, name: name)
                device.gpsString = jsonDic["gpsString"] as? String
                device.alarmOn = (jsonDic["alarmOn"] as? Bool)!
                device.alarmShouldBeOn = (jsonDic["alarmShouldBeOn"] as? Bool)!
                device.untrigger = (jsonDic["untrigger"] as? Bool)!
                device.state = (jsonDic["state"] as? Int).map {DeviceState(rawValue: $0) }!

                if let lastUpdateString = jsonDic["lastUpdate"] {
                    let dateFormatter = DateFormatter()
                    dateFormatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSSZ"
```

```
                device.lastUpdate = dateFormatter.date(from: lastUpdateString as! String)


            }
            completion(device,nil)
        } else {
            completion(nil,DeviceFetchError.unknown)
        }

//          if let data = response.data, let utf8Text = String(data: data, encoding: .utf8) {
//              print("Data: \(utf8Text)") // original server data as UTF8 string
//          }
    }
  }

  func updateDevice(device : Device, completion:@escaping((Device?,DeviceFetchError?)->())) {

    let userToken = UserManager.sharedInstance.user?.token

    let parameters: Parameters = [
        "token" : userToken!,
        "name" : (device.name)!,
        "alarmShouldBeOn" : device.alarmShouldBeOn,
        "untrigger" : device.untrigger
    ]

    Alamofire.request(GuardServiceRoutes.deviceRoute + "/" + device.id, method: .put, parameters: parameters).responseJSON {
response in

        if let json = response.result.value {
            var jsonDic = json as! [String : Any]

            if(response.response?.statusCode == 401) {
                completion(nil,DeviceFetchError.expiredToken)
                return
            }
            jsonDic = jsonDic["device"] as! [String : Any]
            let _id  = jsonDic["_id"] as! String
            let name = jsonDic["name"] as? String
            let device = Device(id: _id, name: name)
            device.gpsString = jsonDic["gpsString"] as? String
            device.alarmOn = (jsonDic["alarmOn"] as? Bool)!
            device.alarmShouldBeOn = (jsonDic["alarmShouldBeOn"] as? Bool)!
            device.untrigger = (jsonDic["untrigger"] as? Bool)!
            device.state = (jsonDic["state"] as? Int).map {DeviceState(rawValue: $0) }!

            if let lastUpdateString = jsonDic["lastUpdate"] {
                let dateFormatter = DateFormatter()
                dateFormatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSSZ"
                device.lastUpdate = dateFormatter.date(from: lastUpdateString as! String)

            }
            completion(device,nil)
        } else {
            completion(nil,DeviceFetchError.unknown)
        }

    }
  }
```

```
}

//
//  Device.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 14/10/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
//
import Foundation
import CoreLocation

enum DeviceState: Int {
   case Active, Pairing, Triggered, Proximity
}


class Device {

   var id : String
   var name : String?
   var gpsString : String?
   var alarmOn = false
   var alarmShouldBeOn = false
   var untrigger = false
   var lastUpdate : Date?
   var state : DeviceState?
   var peripheralIdentifier : UUID?

   var location : CLLocation? {
      get {
         guard self.gpsString != nil else {
            return nil
         }

         if let components = self.gpsString?.components(separatedBy: ",") {

            if components.count >= 2 {
               let latitudeString = components[0].replacingOccurrences(of: " ", with: "")
               let longitudeString = components[1].replacingOccurrences(of: " ", with: "")


               let latitudeDouble = Double(latitudeString)
               let longitudeDouble = Double(longitudeString)



               if (latitudeDouble != nil && longitudeDouble != nil) {

                  if latitudeDouble == 0.0 || longitudeDouble == 0.0 {
                     return nil
                  }

                  return CLLocation(latitude: latitudeDouble!, longitude: longitudeDouble!)
               }
            }
         }

         return nil
      }
```

```
    }

    init(id: String, name: String?) {
        self.id = id
        self.name = name
    }


}


//
//  AppDelegate.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 23/08/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
//

import UIKit
import CoreData
import UserNotifications
import CoreBluetooth
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, BluetoothManagerDelegate {


    func serialDidChangeState() {
        if BluetoothManager.sharedInstance.centralManager.state == .poweredOn {
            BluetoothManager.sharedInstance.startScan()
        }
    }

    func serialDidDisconnect(_ peripheral: CBPeripheral, error: NSError?) {
        //Notificacao saiu de perto?
        let deviceListViewController = self.window?.rootViewController?.childViewControllers[0] as! DeviceListViewController
            let devices = deviceListViewController.devices
            for device in devices {
                if device.peripheralIdentifier  == peripheral.identifier  {
                    let content = UNMutableNotificationContent()

                    content.title = "□ Device left your surroundings"
                    content.body =  "\(device.name!)'s alarm is now active"
                    content.sound = UNNotificationSound.default()

                    // Deliver the notification in five seconds.
                    let trigger = UNTimeIntervalNotificationTrigger.init(timeInterval: 1, repeats: false)
                    let request = UNNotificationRequest.init(identifier: "left_surroundings", content: content, trigger: trigger)

                    // Schedule the notification.
                    let center = UNUserNotificationCenter.current()
                    center.add(request) { (error) in
                        print(error)
                    }
            }
        }
        BluetoothManager.sharedInstance.startScan()
    }

    func serialDidDiscoverPeripheral(_ peripheral: CBPeripheral, RSSI: NSNumber?) {
```

```
        if peripheral.name == "HMSoft" {
            BluetoothManager.sharedInstance.connectToPeripheral(peripheral)
        }
    }

    func serialDidReceiveString(_ message: String) {

        let deviceListViewController = self.window?.rootViewController?.childViewControllers[0] as! DeviceListViewController

        DispatchQueue.main.async {
            let devices = deviceListViewController.devices
            for device in devices {
                if device.id.contains(message) {
                    device.peripheralIdentifier = BluetoothManager.sharedInstance.connectedPeripheral?.identifier
                    BluetoothManager.sharedInstance.sendMessageToDevice("pong")
                }
            }
        }
    }

    var window: UIWindow?


    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey:
Any]?) -> Bool {
        // Override point for customization after application launch.

        BluetoothManager.sharedInstance.delegate = self
        return true
    }

    func applicationWillResignActive(_ application: UIApplication) {
        // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary
interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to
the background state.
        // Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering callbacks. Games should use this
method to pause the game.
    }

    func applicationDidEnterBackground(_ application: UIApplication) {
        // Use this method to release shared resources, save user data, invalidate timers, and store enough application state
information to restore your application to its current state in case it is terminated later.
        // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user
quits.
    }

    func applicationWillEnterForeground(_ application: UIApplication) {
        // Called as part of the transition from the background to the active state; here you can undo many of the changes made on
entering the background.
    }

    func applicationDidBecomeActive(_ application: UIApplication) {

        application.applicationIconBadgeNumber = 0

        let bluetoothIsOn = BluetoothManager.sharedInstance.isPoweredOn


        // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in
the background, optionally refresh the user interface.
    }
```

```
    func applicationWillTerminate(_ application: UIApplication) {
        // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
    }


    lazy var persistentContainer: NSPersistentContainer = {
        /*
         The persistent container for the application. This implementation
         creates and returns a container, having loaded the store for the
         application to it. This property is optional since there are legitimate
         error conditions that could cause the creation of the store to fail.
         */
        let container = NSPersistentContainer(name: "lilguard")
        container.loadPersistentStores(completionHandler: { (storeDescription, error) in
            if let error = error as NSError? {
                // Replace this implementation with code to handle the error appropriately.
                // fatalError() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.

                /*
                 Typical reasons for an error here include:
                 * The parent directory does not exist, cannot be created, or disallows writing.
                 * The persistent store is not accessible, due to permissions or data protection when the device is locked.
                 * The device is out of space.
                 * The store could not be migrated to the current model version.
                 Check the error message to determine what the actual problem was.
                 */
                fatalError("Unresolved error \(error), \(error.userInfo)")
            }
        })
        return container
    }()

    // MARK: - Core Data Saving support

    func saveContext () {
        let context = persistentContainer.viewContext
        if context.hasChanges {
            do {
                try context.save()
            } catch {
                // Replace this implementation with code to handle the error appropriately.
                // fatalError() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.
                let nserror = error as NSError
                fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
            }
        }
    }

    static var moc : NSManagedObjectContext{
        get{
        let delegate = UIApplication.shared.delegate as! AppDelegate
        return delegate.persistentContainer.viewContext
        }
    }

    func registerForPushNotifications() {
        UNUserNotificationCenter.current().requestAuthorization(options: [.alert,.sound,.badge]) { (granted, error) in
```

```
            guard granted else { return }
            self.getNotificationSettings()
        }
    }

    func getNotificationSettings() {
        UNUserNotificationCenter.current().getNotificationSettings { (settings) in
            print("Notification settings: \(settings)")
            guard settings.authorizationStatus == .authorized else { return }
            UIApplication.shared.registerForRemoteNotifications()
        }
    }

    func application(_ application: UIApplication,
                     didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
        let tokenParts = deviceToken.map { data -> String in
            return String(format: "%02.2hhx", data)
        }

        let token = tokenParts.joined()
        UserManager().sendNotificationToken(token: token)
        print("Device Token: \(token)")
    }



}

    //
    //  DeviceListViewController.swift
    //  lilguard-app
    //
    //  Created by Christian Becker Pepino on 27/09/17.
    //  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
    //
    import UIKit
    import CoreLocation
    import MapKit
    class DeviceListViewController : UIViewController, UICollectionViewDelegate, UICollectionViewDataSource,
UICollectionViewDelegateFlowLayout, CLLocationManagerDelegate {

        @IBOutlet weak var collectionView: UICollectionView!
        let deviceFetcher = DeviceManager()
        var devices = [Device]()

        let locationManager = CLLocationManager()
        var currentLocation : CLLocation?

        @IBOutlet weak var welcomeLabel: UILabel!
        @IBOutlet weak var activityIndicator: UIActivityIndicatorView!

        var interfaceTimer : Timer?
        var modelTimer : Timer?


        override func viewDidLoad() {
            self.interfaceTimer = Timer(timeInterval: 1.0, target: self, selector: "refreshUI", userInfo: nil, repeats: true)
            self.modelTimer = Timer(timeInterval: 5.0, target: self, selector: "refresh", userInfo: nil, repeats: true)

            guard let user = UserManager.sharedInstance.user else{
                self.performSegue(withIdentifier: "login", sender: self)
```

```
            return
        }

        locationManager.delegate = self
        locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters
        locationManager.distanceFilter = 10.0

        RunLoop.main.add(self.interfaceTimer!, forMode: RunLoopMode.defaultRunLoopMode)
        RunLoop.main.add(self.modelTimer!, forMode: RunLoopMode.defaultRunLoopMode)



    }

    override func viewWillAppear(_ animated: Bool) {

        UIApplication.shared.statusBarStyle = .default


        if let user = UserManager.sharedInstance.user {
            activityIndicator.startAnimating()
            self.locationManager.requestWhenInUseAuthorization()
            self.locationManager.startUpdatingLocation()
            refresh()

        }
        self.navigationController?.setNavigationBarHidden(true, animated: false)
    }
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        return devices.count + 1
    }

    func numberOfSections(in collectionView: UICollectionView) -> Int {
        return 1
    }


    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {

        var cell : UICollectionViewCell?

        if(indexPath.row == (devices.count)) {
            cell = self.collectionView.dequeueReusableCell(withReuseIdentifier: "AddDeviceCell", for: indexPath)

        } else {
            cell = self.collectionView.dequeueReusableCell(withReuseIdentifier: "DeviceCell", for: indexPath)

            let device = devices[indexPath.row]
            let deviceNameLabel = cell?.viewWithTag(1) as! UILabel
            let deviceLastUpdateLabel = cell?.viewWithTag(5) as! UILabel
            let distanceLabel = cell?.viewWithTag(2) as! UILabel
            let alarmLabel = cell?.viewWithTag(3) as! UILabel
            let alarmLayer = cell?.viewWithTag(6) as! AlarmLayerView

            if (currentLocation != nil && device.location != nil) {
                let distance : Double = (device.location?.distance(from: currentLocation!))!
                let distanceFormatter = MKDistanceFormatter()
                distanceFormatter.unitStyle = .abbreviated
                distanceLabel.text = "distance: \(distanceFormatter.string(fromDistance: distance))"
            } else {
                distanceLabel.text = "distance: -"
```

```
            }

            if device.state == DeviceState.Proximity {
                distanceLabel.text = "distance: nearby"
            }


            if device.state == DeviceState.Triggered {
                alarmLabel.text = "alarm: triggered"
            } else {
                if device.alarmOn {
                    alarmLabel.text = "alarm: on"
                } else {
                    alarmLabel.text = "alarm: off"
                }
            }

            if device.state == DeviceState.Triggered {
                UIView.animate(withDuration: 1.5, delay: 0.5, options: [.curveEaseOut], animations: {
                    alarmLayer.isHidden = !alarmLayer.isHidden
                }, completion: nil)
            } else {
                alarmLayer.isHidden = true
            }

            deviceNameLabel.text = device.name

            let hourMinuteSecond: Set<Calendar.Component> = [.hour, .minute, .second]
            if let lastUpdatedate = devices[indexPath.row].lastUpdate {
                let difference = NSCalendar.current.dateComponents(hourMinuteSecond, from: (devices[indexPath.row].lastUpdate)!,
to: Date())

                if (difference.hour!>0) {
                    deviceLastUpdateLabel.text = "last update: \(difference.hour!)h \(difference.minute!)m \(difference.second!)s ago"
                } else {
                    if (difference.minute!>0){
                        deviceLastUpdateLabel.text = "last update: \(difference.minute!)m \(difference.second!)s ago"
                    }else {
                        if (difference.second!>0){
                            deviceLastUpdateLabel.text = "last update: \(difference.second!)s ago"
                        }
                    }
                }
            }
        }



        cell!.contentView.layer.cornerRadius = 10.0
        cell!.contentView.layer.borderWidth = 1.0
        cell!.contentView.layer.borderColor = UIColor.clear.cgColor
        cell!.contentView.layer.masksToBounds = true

        cell!.layer.shadowColor = UIColor.lightGray.cgColor
        cell!.layer.cornerRadius = 10.0
        cell!.layer.shadowOffset = CGSize(width: 0, height: 2.0)
        cell!.layer.shadowRadius = 2.0
        cell!.layer.shadowOpacity = 1.0
        cell!.layer.masksToBounds = false
```

```
            cell!.layer.shadowPath = UIBezierPath(roundedRect: cell!.bounds, cornerRadius:
cell!.contentView.layer.cornerRadius).cgPath

            cell!.layer.borderWidth=1.0;
            cell!.layer.borderColor = UIColor(displayP3Red: 151/255, green: 151/255, blue: 151/255, alpha: 0.13).cgColor

            return cell!
        }
        func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt
indexPath: IndexPath) -> CGSize {

            //22 de margem lateral, 20 entre os caras
            let edge = self.view.frame.width/2 - 32
            return CGSize(width: edge, height: edge);
        }

        override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
            if segue.identifier == "Device Details" {
                let cell = sender as! UICollectionViewCell
                let deviceViewController = segue.destination as! DeviceViewController
                let index = self.collectionView.indexPath(for: cell)?.row
                deviceViewController.device = self.devices[index!]
                deviceViewController.currentLocation = self.currentLocation
            }
        }

        //MARK: - Location Manager Delegate
        func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
            self.currentLocation = locations.last
        }

        @objc func refresh() {
            DispatchQueue.main.async {
                if let user = UserManager.sharedInstance.user {
                    self.deviceFetcher.fetchDevices(){
                        devices, error in
                        if(error == nil) {
                            DispatchQueue.main.async {
                                self.devices = devices!
                            }
                            self.activityIndicator.stopAnimating()

                        } else {
                            switch error! {
                            case .expiredToken:
                                self.performSegue(withIdentifier: "login", sender: self)
                                break
                            case .unknown:
                                print("An unknown error occured.");
                                break
                            }
                        }

                    }
                }
            }
        }

        @objc func refreshUI() {
            DispatchQueue.main.async {
```

```
            if let user = UserManager.sharedInstance.user {
                self.collectionView.reloadData()
                for device in self.devices {
                    if device.state == DeviceState.Triggered {
                        self.welcomeLabel.text = "The alarm was triggered!"
                    } else {
                        if let userFirstName = user.name?.components(separatedBy:" ").first {
                            self.welcomeLabel.text = "Welcome back \(userFirstName), things look alright."
                        }
                    }
                }
            }
        }

    }


//
//  DeviceViewController.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 07/10/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
//

import UIKit
import MapKit
import CoreLocation
class DeviceViewController: UIViewController, UITextFieldDelegate, CLLocationManagerDelegate{

    @IBOutlet weak var deviceNameTextField: UITextField!
    @IBOutlet weak var mapView: MKMapView!
    @IBOutlet weak var alarmLabel: UILabel!
    @IBOutlet weak var waitingForDeviceLabel: UILabel!
    @IBOutlet weak var distanceLabel: UILabel!
    @IBOutlet weak var lastStatusLabel: UILabel!
    @IBOutlet weak var alarmButton: UIButton!

    @IBOutlet weak var locationUnavailableLabel: UILabel!

    var device : Device?
    var currentLocation : CLLocation?
    let locationManager = CLLocationManager()

    let deviceManager = DeviceManager()

    var modelTimer : Timer?
    var interfaceTimer : Timer?


    override func viewDidLoad() {
        super.viewDidLoad()
        self.navigationController?.navigationBar.setBackgroundImage(UIImage(), for: .default)
        self.navigationController?.navigationBar.shadowImage = UIImage()
        self.navigationController?.navigationBar.isTranslucent = true
        self.navigationController?.view.backgroundColor = .clear

        let tap: UITapGestureRecognizer = UITapGestureRecognizer(
            target: self,
            action: #selector(dismissKeyboard))
```

```
        view.addGestureRecognizer(tap)

        locationManager.delegate = self
        locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters
        locationManager.distanceFilter = 10.0

        self.locationManager.requestWhenInUseAuthorization()
        self.locationManager.startUpdatingLocation()

        self.modelTimer = Timer(timeInterval: 5.0, target: self, selector: #selector(DeviceViewController.refreshModel), userInfo: nil,
repeats: true)
        self.interfaceTimer = Timer(timeInterval: 1.0, target: self, selector: #selector(DeviceViewController.refresh), userInfo: nil,
repeats: true)
        self.refresh()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        self.navigationController?.setNavigationBarHidden(false, animated: true)
        RunLoop.main.add(self.modelTimer!, forMode: RunLoopMode.defaultRunLoopMode)
        RunLoop.main.add(self.interfaceTimer!, forMode: RunLoopMode.defaultRunLoopMode)


    }

    override func viewWillDisappear(_ animated: Bool) {
        self.modelTimer?.invalidate()
        self.interfaceTimer?.invalidate()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    @objc func refreshModel() {

        deviceManager.fetchDeviceWithId(deviceId: device!.id) { (device, error) in
            DispatchQueue.main.async {


                if device != nil {
                    self.device = device
                }
            }
        }
    }

    @objc func refresh() {
        self.refreshUI()
        self.updateMap()
    }

    func refreshUI() {

        if (device?.state == DeviceState.Triggered) {
            alarmLabel.text = "alarm: triggered"
        } else {
            if (device?.alarmOn)! {
```

```
        alarmLabel.text = "alarm: on"
    } else {
        alarmLabel.text = "alarm: off"
    }
}

if (currentLocation != nil && device?.location != nil && device?.state != DeviceState.Proximity)    {
    let distance : Double = (device!.location?.distance(from: currentLocation!))!
    let distanceFormatter = MKDistanceFormatter()
    distanceFormatter.unitStyle = .abbreviated
    distanceLabel.text = "distance: \(distanceFormatter.string(fromDistance: distance))"
} else {
    if device?.state == DeviceState.Proximity {
        distanceLabel.text = "distance: nearby"
    } else {
        distanceLabel.text = "distance: -"
    }
}

if(device?.state != DeviceState.Triggered) {
    if device?.alarmShouldBeOn != device?.alarmOn {
        waitingForDeviceLabel.isHidden = false
        alarmButton.alpha = 0.5
    } else {
        waitingForDeviceLabel.isHidden = true
        alarmButton.alpha = 1

    }
} else {
    if device?.untrigger == true {
        waitingForDeviceLabel.isHidden = false
        alarmButton.alpha = 0.5
    } else {
        waitingForDeviceLabel.isHidden = true
        alarmButton.alpha = 1
    }

}

if (device?.alarmOn)! {
    self.alarmButton.setTitle("Deactivate alarm", for: .normal)
} else {
    self.alarmButton.setTitle("Activate alarm", for: .normal)
}

let hourMinuteSecond: Set<Calendar.Component> = [.hour, .minute, .second]
if let lastUpdatedate = device?.lastUpdate {
    let difference = NSCalendar.current.dateComponents(hourMinuteSecond, from: (device?.lastUpdate)!, to: Date())

    if (difference.hour!>0) {
        lastStatusLabel.text = "last update: \(difference.hour!)h \(difference.minute!)m \(difference.second!)s ago"
    } else {
        if (difference.minute!>0){
            lastStatusLabel.text = "last update: \(difference.minute!)m \(difference.second!)s ago"
        }else {
            if (difference.second!>0){
                lastStatusLabel.text = "last update: \(difference.second!)s ago"
            }
        }
    }
```

```
    }


    if !deviceNameTextField.isEditing {
        deviceNameTextField.text = device?.name
    }
}

func updateMap() {

    var location : CLLocation?

    if (device?.state != DeviceState.Proximity) {
        location = device?.location
    } else {
        location = currentLocation
    }

    if location != nil {

        var annotation : MKPointAnnotation

        if mapView.annotations.first != nil {
            annotation = (mapView.annotations.first as? MKPointAnnotation)!
            UIView.animate(withDuration: 0.300, animations: {
                annotation.coordinate = (location?.coordinate)!
            })
            let viewRegion = MKCoordinateRegionMakeWithDistance((location?.coordinate)!, 500, 500)
            self.mapView.setRegion(viewRegion, animated: true)
            mapView.setCenter((location?.coordinate)!, animated: true)

        } else {
            annotation = MKPointAnnotation();
            annotation.coordinate = (location?.coordinate)!
            mapView.addAnnotation(annotation)
            let viewRegion = MKCoordinateRegionMakeWithDistance((location?.coordinate)!, 500, 500)
            self.mapView.setRegion(viewRegion, animated: false)
            mapView.setCenter((location?.coordinate)!, animated: false)
        }

        locationUnavailableLabel.isHidden = true
        self.mapView.alpha = 1
    } else {
        locationUnavailableLabel.isHidden = false
        self.mapView.alpha = 0.5
    }

}

@IBAction func alarmButtonEvent(_ sender: Any) {
    if (device?.state != DeviceState.Triggered) {
        device?.alarmShouldBeOn = !(device?.alarmShouldBeOn)!

    } else {
        device?.untrigger = !(device?.untrigger)!
    }

    deviceManager.updateDevice(device: device!) { (device, error) in
        if(device != nil) {
            self.device = device
        }
```

```
        }
    }
    @IBAction func unpairButtonEvent(_ sender: Any) {
        let alert = UIAlertController(title: "Alert", message: "Are you sure you want to unpair your device?", preferredStyle: .alert)

        alert.addAction(UIAlertAction(title: "Cancel", style:.default) { action in
        })

        alert.addAction(UIAlertAction(title: "Unpair", style: .destructive, handler: { (action) in
            self.deviceManager.unpair(deviceId: (self.device?.id)!, success: {
                self.navigationController?.popViewController(animated: true)
            }, failure: { _  in
                let alert = UIAlertController(title: "Error", message: "Could not unpair device", preferredStyle: .alert)
                self.present(alert, animated: true, completion: nil)

            })
        }))

        self.present(alert, animated: true, completion: nil)

    }

    @objc func dismissKeyboard() {
        deviceNameTextField.resignFirstResponder()
    }

    // MARK: - Text Field Delegate
    func textFieldDidEndEditing(_ textField: UITextField) {

        if(textField.text != "") {
            device?.name = textField.text
            deviceManager.updateDevice(device: device!) { (device, error) in
                if(device != nil) {
                    self.device = device
                    self.refresh()
                }
            }

        } else {
            textField.text = device?.name
        }
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        return true
    }

    //MARK: - Location Manager Delegate
    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
        self.currentLocation = locations.last
    }

}

//
//  LoginViewController.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 12/10/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
```

```
//

import UIKit

class LoginViewController : UIViewController, RegisterViewControllerDelegate,UITextFieldDelegate{


    @IBOutlet weak var userNameTextField: UITextField!
    @IBOutlet weak var passwordTextField: UITextField!
    @IBOutlet weak var scrollView: UIScrollView!

    @IBOutlet weak var fieldsView: UIView!

    override func viewDidLoad() {
        registerKeyboardNotifications()

        let tap: UITapGestureRecognizer = UITapGestureRecognizer(
            target: self,
            action: #selector(dismissKeyboard))

        view.addGestureRecognizer(tap)

        self.navigationController?.navigationBar.setBackgroundImage(UIImage(), for: .default)
        self.navigationController?.navigationBar.shadowImage = UIImage()
        self.navigationController?.navigationBar.isTranslucent = true
        self.navigationController?.view.backgroundColor = .clear

    }
    fileprivate func registerKeyboardNotifications() {
        NotificationCenter.default.addObserver(self, selector: #selector(LoginViewController.keyboardWillShow), name:
NSNotification.Name.UIKeyboardWillShow, object: nil)
        NotificationCenter.default.addObserver(self, selector: #selector(LoginViewController.keyboardWillHide), name:
NSNotification.Name.UIKeyboardWillHide, object: nil)
    }

    @objc fileprivate func deRegisterKeyboardNotifications() {
        NotificationCenter.default.removeObserver(self, name: .UIKeyboardWillShow, object: nil)
        NotificationCenter.default.removeObserver(self, name: .UIKeyboardDidHide, object: nil)
    }
    override func viewWillAppear(_ animated: Bool) {
        self.navigationController?.setNavigationBarHidden(true, animated: false)
        UIApplication.shared.statusBarStyle = .lightContent
    }
    override func viewWillDisappear(_ animated: Bool) {
        self.navigationController?.setNavigationBarHidden(false, animated: true)
    }

    @IBAction func loginButtonEvent(_ sender: UIButton?) {

        UserManager.sharedInstance.login(userName: userNameTextField.text!, password: passwordTextField.text!, success: {
            let delegate = UIApplication.shared.delegate as! AppDelegate
            delegate.registerForPushNotifications()
            self.dismiss(animated: true, completion: nil)
        }, failure: { message in

            let alert = UIAlertController(title: "Error", message: message, preferredStyle: .alert)
            alert.addAction(UIAlertAction(title: "Ok", style: .cancel) { action in
            })
            self.present(alert, animated: true, completion: nil)
        })
    }
```

```
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "register" {
            let registerViewController = segue.destination as! RegisterViewController
            registerViewController.delegate = self
        }
    }
    func registerViewController(viewController: RegisterViewController, didRegisterUserName userName: String) {
        self.userNameTextField.text  = userName
        self.navigationController?.popToViewController(self, animated: true)
    }

    @objc fileprivate func keyboardWillShow(notification: NSNotification) {
        // this method will get called even if a system generated alert with keyboard appears over the current VC.
        var activeTextField : UITextField

        if (userNameTextField.isEditing){
            activeTextField = userNameTextField
        } else {
            activeTextField = passwordTextField
        }

            let info: NSDictionary = notification.userInfo! as NSDictionary
            let value: NSValue = info.value(forKey: UIKeyboardFrameEndUserInfoKey) as! NSValue
            let keyboardSize: CGSize = value.cgRectValue.size
            let contentInsets: UIEdgeInsets = UIEdgeInsetsMake(0.0, 0.0, keyboardSize.height, 0.0)
            scrollView.contentInset = contentInsets
            scrollView.scrollIndicatorInsets = contentInsets

            // If active text field is hidden by keyboard, scroll it so it's visible
            // Your app might not need or want this behavior.
            var aRect: CGRect = self.view.frame
            aRect.size.height -= keyboardSize.height
            let rect = CGRect(x: fieldsView.frame.origin.x, y: fieldsView.frame.origin.y + 110, width: fieldsView.frame.size.width, height:
fieldsView.frame.size.height)
                scrollView.scrollRectToVisible(rect, animated:true)

    }

    @objc func keyboardWillHide(notification: NSNotification) {
        let contentInsets: UIEdgeInsets = .zero
        scrollView.contentInset = contentInsets
        scrollView.scrollIndicatorInsets = contentInsets
    }

    @objc func dismissKeyboard() {
        view.endEditing(true)
    }

    //MARK: - UITextFieldDelegate

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        if textField == userNameTextField {
            passwordTextField.becomeFirstResponder()
            return false
        } else {
            self.loginButtonEvent(nil)
            return true
        }
    }
```

```
}
//
//  RegisterViewController.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 18/11/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
//

import UIKit

protocol RegisterViewControllerDelegate {
    func registerViewController(viewController:RegisterViewController, didRegisterUserName userName: String)
}

class RegisterViewController : UIViewController, UITextFieldDelegate    {

    @IBOutlet weak var usernameTextField: UITextField!
    @IBOutlet weak var passwordTextField: UITextField!
    @IBOutlet weak var nameTextField: UITextField!
    @IBOutlet weak var registerButton: UIButton!

    @IBOutlet weak var scrollView: UIScrollView!
    @IBOutlet weak var fieldsView: UIView!
    var activeTextField : UITextField?
    var delegate : RegisterViewControllerDelegate?


    override var preferredStatusBarStyle: UIStatusBarStyle {
        return .lightContent
    }

    override func viewDidLoad() {

        let tap: UITapGestureRecognizer = UITapGestureRecognizer(
            target: self,
            action: #selector(dismissKeyboard))

        view.addGestureRecognizer(tap)

        NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillShow), name: .UIKeyboardWillShow, object: nil)
        NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillHide), name: .UIKeyboardWillHide, object: nil)

    }

    override func viewWillDisappear(_ animated: Bool) {

        NotificationCenter.default.removeObserver(self, name: .UIKeyboardWillShow, object: nil)
        NotificationCenter.default.removeObserver(self, name: .UIKeyboardWillHide, object: nil)
    }

    override func viewWillAppear(_ animated: Bool) {
        UIApplication.shared.statusBarStyle = .lightContent
    }

    func textFieldDidBeginEditing(_ textField: UITextField) {
        activeTextField = textField
    }


    @IBAction func registerButtonEvent(_ sender: UIButton) {
```

```
        UIApplication.shared.isNetworkActivityIndicatorVisible = true
        registerButton.isUserInteractionEnabled = false
        UserManager.sharedInstance.register(userName: self.usernameTextField.text!, password: self.passwordTextField.text!, name:
self.nameTextField.text!, success: {userName in
            self.delegate?.registerViewController(viewController: self, didRegisterUserName: userName)
            self.registerButton.isUserInteractionEnabled = true
            UIApplication.shared.isNetworkActivityIndicatorVisible = false

        },failure:  { (message) in
            let alert = UIAlertController(title: "Error", message: message, preferredStyle: .alert)
            alert.addAction(UIAlertAction(title: "Ok", style: .cancel) { action in
            })
            self.present(alert, animated: true, completion: nil)
            self.registerButton.isUserInteractionEnabled = true
            UIApplication.shared.isNetworkActivityIndicatorVisible = false


        })
    }

    @objc fileprivate func keyboardWillShow(notification: NSNotification) {
        // this method will get called even if a system generated alert with keyboard appears over the current VC.


        let info: NSDictionary = notification.userInfo! as NSDictionary
        let value: NSValue = info.value(forKey: UIKeyboardFrameEndUserInfoKey) as! NSValue
        let keyboardSize: CGSize = value.cgRectValue.size
        let contentInsets: UIEdgeInsets = UIEdgeInsetsMake(0.0, 0.0, keyboardSize.height, 0.0)
        scrollView.contentInset = contentInsets
        scrollView.scrollIndicatorInsets = contentInsets

        // If active text field is hidden by keyboard, scroll it so it's visible
        // Your app might not need or want this behavior.
        var aRect: CGRect = self.view.frame
        aRect.size.height -= keyboardSize.height
        let rect = CGRect(x: fieldsView.frame.origin.x, y: fieldsView.frame.origin.y + 110, width: fieldsView.frame.size.width, height:
fieldsView.frame.size.height)
        scrollView.scrollRectToVisible(rect, animated:true)

    }

    @objc func keyboardWillHide(notification:NSNotification) {

        if let keyboardSize = (notification.userInfo?[UIKeyboardFrameBeginUserInfoKey] as? NSValue)?.cgRectValue {
            self.scrollView.frame = CGRect(x: self.scrollView.frame.origin.x, y: self.scrollView.frame.origin.y, width:
self.scrollView.frame.size.width, height: self.scrollView.frame.size.height + keyboardSize.height)
        }

    }

    @objc func dismissKeyboard() {
        view.endEditing(true)
    }
}

//
//  PairViewController.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 19/11/17.
//  Copyright © 2017 Christian Becker Pepino and Guilherme Dias. All rights reserved.
```

```
//

import UIKit
import CoreBluetooth
class PairViewController: UIViewController, BluetoothManagerDelegate {


    @IBOutlet weak var deviceIdTextField: UITextField!
    @IBOutlet weak var activityIndicator: UIActivityIndicatorView!
    @IBOutlet weak var statusLabel: UILabel!

    var fetchedDeviceId : String?

    override func viewDidLoad() {
        super.viewDidLoad()
        BluetoothManager.sharedInstance.delegate = self
        BluetoothManager.sharedInstance.startScan()
    }

    override func viewWillAppear(_ animated: Bool) {
        UIApplication.shared.statusBarStyle = .lightContent
    }
    override func viewDidDisappear(_ animated: Bool) {
        BluetoothManager.sharedInstance.startScan()
        BluetoothManager.sharedInstance.delegate = UIApplication.shared.delegate as! AppDelegate
    }

    @IBAction func pairButtonEvent(_ sender: Any) {

        var deviceIdString : String?

        if self.deviceIdTextField.text != nil && self.deviceIdTextField.text != "" {
            deviceIdString = deviceIdTextField.text!
        } else {
            deviceIdString = fetchedDeviceId
        }

        guard deviceIdString != nil else {
            return
        }

        activityIndicator.startAnimating()

        self.deviceIdTextField.resignFirstResponder()

        DeviceManager.sharedInstace.pair(deviceId: deviceIdString!, success: {
            self.activityIndicator.stopAnimating()
            let alert = UIAlertController(title: "Device Paired", message: nil, preferredStyle: .alert)
            alert.addAction(UIAlertAction(title: "Ok", style: .cancel) { action in
                DispatchQueue.main.async {
                    self.dismiss(animated: true, completion: nil)
                }
            })

            self.present(alert, animated: true, completion: nil)

            BluetoothManager.sharedInstance.sendMessageToDevice("f")


        }) { (message) in
            let alert = UIAlertController(title: "Request error", message: message, preferredStyle: .alert)
```

```
        alert.addAction(UIAlertAction(title: "Ok", style: .cancel) { action in
        })
        self.present(alert, animated: true, completion: nil)
    }
}

@IBAction func cancelButtonEvent(_ sender: Any) {
    //cancelar a busca bluetooth e fechar a tela
    self.dismiss(animated: true, completion: nil)
}

@IBAction func pairManuallyButtonEvent(_ sender: Any) {
    self.deviceIdTextField.isHidden = false
}

override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    // Dispose of any resources that can be recreated.
}

//MARK: - BluetoothManagerDelegate
func serialDidChangeState() {
    if BluetoothManager.sharedInstance.centralManager.state == .poweredOff {
        let alert = UIAlertController(title: "Bluetooth Off", message:"", preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "Ok", style: .cancel) { action in
        })
        self.present(alert, animated: true, completion: nil)
    }
}

func serialDidDisconnect(_ peripheral: CBPeripheral, error: NSError?) {
    let alert = UIAlertController(title: "Disconnected", message:"", preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "Ok", style: .cancel) { action in
    })
    self.present(alert, animated: true, completion: nil)
}

func serialDidReceiveString(_ message: String) {
    if message.contains("1:") {
        fetchedDeviceId = message.components(separatedBy:":").last
    }
    if message.contains("2:") {
        self.statusLabel.text = "Tracker found"
        fetchedDeviceId = fetchedDeviceId! + message.components(separatedBy:":").last!
    }
}

func serialDidConnect(_ peripheral: CBPeripheral) {
    activityIndicator.stopAnimating()
}

func serialDidDiscoverPeripheral(_ peripheral: CBPeripheral, RSSI: NSNumber?) {
    if peripheral.name == "HMSoft" {
        BluetoothManager.sharedInstance.stopScan()
        BluetoothManager.sharedInstance.connectToPeripheral(peripheral)
    }
}

func serialIsReady(_ peripheral: CBPeripheral) {
    BluetoothManager.sharedInstance.sendMessageToDevice("h")
```

```
    }

}

//
//  AlarmLayerView.swift
//  lilguard-app
//
//  Created by Christian Becker Pepino on 20/01/18.
//  Copyright © 2018 Christian Becker Pepino. All rights reserved.
//

import UIKit

class AlarmLayerView: UIView {

    private var isBlinking = false

    func blink() {
        if isBlinking == false {
            isBlinking = true
            UIView.animate(withDuration: 1.0, animations: {
                self.alpha = 1.0
            }, completion : {
                finished in
                UIView.animate(withDuration: 1.0, animations: {
                    self.alpha = 0.0
                    self.isBlinking = false
                })
            })
        }
    }

    func stopBlink() {
        isBlinking = false
        self.alpha = 0.0
        self.layer.removeAllAnimations()
    }
    /*
    // Only override draw() if you perform custom drawing.
    // An empty implementation adversely affects performance during animation.
    override func draw(_ rect: CGRect) {
        // Drawing code
    }
    */

}
```

APÊNDICE B – CÓDIGO FONTE DO SERVIDOR

```
var express     = require('express');
var app         = express();
var bodyParser  = require('body-parser');
var morgan      = require('morgan');
var mongoose    = require('mongoose');
var apn = require('apn');
var jwt    = require('jsonwebtoken');
var config = require('./config');
var User   = require('./app/models/user');
var Device = require('./app/models/device');

var port = process.env.PORT || 3000;


mongoose.connect(config.database,function(err) {
   if (err) {
      console.log(err);
   } else {
      console.log('Connected');
   }
});

app.set('superSecret', config.secret); // secret variable

app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(morgan('dev'));


var options = {
   token: {
      key: "AuthKey_6ETX75T8C5.p8",
      keyId: "6ETX75T8C5",
      teamId: "6WXQMFM3EL"
   },
   production: false
};

var apnProvider = new apn.Provider(options);

// =======================
// routes ================
// =======================
// basic route

app.get('/', function(req, res) {
   res.send('Hello! The API is at http://localhost:' + port + '/api');
});

app.post('/signup', function(req, res) {
   User.findOne({
      userName: req.body.userName
   }, function(err,user) {
      if(err) throw err;
      if (!user){
         var user = new User({
            userName: req.body.userName,
            password: req.body.password,
```

```
            name: req.body.name,
            admin: false
        })

        user.save(function(err){
            if (err) throw err

            res.status(201).send({
                success: true,
                message: 'User created.',
                userName: req.body.userName
            })});
    } else {
        return res.status(409).send({
            success: false,
            message: 'User already exists'
        });
    }
  });
});

app.get('/setup', function(req,res) {
    var user = new User({
        name: 'O Admin',
        userName : 'admin',
        password: 'admin',
        admin : true
    });

    user.save(function(err){
        if(err)    throw err;
        console.log('Admin created successfully');
        res.json({success:true})
    });
});

app.post('/authenticate',function(req,res) {
    User.findOne({
        userName: req.body.userName
    }, function(err,user) {
        if(err) throw err;

        if(!user) {
            res.json({success:false, message:'User doesn\'t exist'});
        } else {
            if(user.password != req.body.password) {
                res.json({success:false, message:'Wrong password'});
            } else {
                var token = jwt.sign(user.toObject(), app.get('superSecret'), {
                    expiresIn : 60*60*24*30, //30 dias
                });

                res.json({
                    success: true,
                    message: 'Here is your token',
                    token: token,
                    user: user
                });
            }
        }
```

```
    });

});

//DEVELOPMENT FUNCTIONS
app.get('/users', function(req, res) {
    User.find({}, function(err, users) {
        res.json(users);
    });
});

app.put('/eraseUsers', function(req,res) {
    User.remove({}, function(err) {
        if(err)
        throw err;
    });
    res.send('All users erased');

});

app.post('/devices',function(req,res){

    var device = new Device({
        alarmOn: false,
        alarmShouldBeOn : true,
        untrigger : false,
        state: 0
    })

    device.save(function(err){
        if(err) throw err;
        return res.status(201).send({
            success:true,
            message:`Device created`,
            device: device
        });
    });
});


// Chamada exclusiva do device, nao do smartphone
app.get('/deviceStatus',function(req,res){
    Device.findOne({
        _id: req.query.deviceId
    }).populate('user').exec(function(err,device) {
        if (err) throw err;
        if(device){
            if(device.state != 2 && req.query.state == 2 && device.alarmShouldBeOn === true) {

                let deviceToken = device.user.notificationToken
                var note = new apn.Notification();
                note.expiry = Math.floor(Date.now() / 1000) + 3600; // Expires 1 hour from now.
                note.badge = 1;
                note.sound = "ping.aiff";
                note.alert =  "\uD83D\uDEA8 Alarm Triggered for " + device.name;
                note.payload = {'messageFrom': 'Pocket Guard'};
                note.topic = "CBP.lilguard-app";

                apnProvider.send(note, deviceToken).then( (result) => {

                    });
```

```
            }
            var deviceShouldUntrigger = device.untrigger;

            if(!(req.query.gpsString === "")){
                device.gpsString = req.query.gpsString;
            }

            device.lastUpdate = new Date();
            device.alarmOn = req.query.alarmOn;
            device.state = req.query.state;
            device.untrigger = false;

            var paired = 0;
            if (device.user) {
                paired = 1;
            }

            device.save(function(err){
                if (err) throw err;
                return res.status(200).send({
                    a: device.alarmShouldBeOn,
                    p: paired,
                    u :deviceShouldUntrigger
                });
            });} else {
            return res.status(404).send({
                success: false,
            });
        }
    });
});


// API ROUTES -------------------
var router = express.Router();


router.use(function(req,res,next) {
    var token = req.body.token || req.query.token || req.headers['x-access-token'];
    if(token) {
        jwt.verify(token, app.get('superSecret'), function(err,decoded) {
            if(err) {
                return res.status(401).send({ success: false, message: 'Failed to authenticate token.' });

            } else {
                req.decoded = decoded;
                next();
            }
        });    }else {
        return res.status(403).send({
            success: false,
            message: 'No token provided.'
        });
    }
});

router.get('/',function(req,res) {
    res.json({message: 'Welcome to the best API on earth!'});
});
```

```
router.put('/pair',function(req,res) {
    User.findOne({
        userName: req.decoded.userName
    },function(err,user){

        if(err) throw err;

        if(user) {
            Device.findOne({
                _id: req.body.deviceId
            },function(err,device){
                if(err) throw err;
                if(device){
                    device.user = user;
                    device.alarmShouldBeOn = true;
                    device.untrigger = false;
                    device.gpsString = "";
                    user.devices.addToSet(device);
                    var deviceNumber = user.devices.length + 1;
                    var deviceString = deviceNumber.toString();
                    device.name = 'device' + deviceString;
                    device.save(function(err){
                        if (err) throw err;
                        user.save(function(err){
                            if (err) throw err
                            return res.status(201).send({
                                success: true,
                                message: 'Paired!'
                            });
                        });

                    });
                }
            });
        }
    });
});

router.put('/unpair',function(req,res) {
    User.findOne({
        userName: req.decoded.userName
    },function(err,user){
        if(err) throw err;
        if(user) {
            Device.findOne({
                _id: req.body.deviceId
            },function(err,device){
                if(err) throw err;
                if(device){
                    device.user = null;
                    user.devices.remove(device);
                    device.save(function(err){
                        if (err) throw err;
                        user.save(function(err){
                            if (err) throw err
                            return res.status(201).send({
                                success: true,
                                message: 'Paired!'
                            });
```

```
                    });

                });
            }
        });
    }
});

router.get('/devices',function(req,res){
    User.findOne({
        userName:req.decoded.userName
    }).populate('devices').exec(function(err,user){
        if(err) throw err;
        if(user) {
            return res.status(200).send({
                devices: user.devices
            });
        }else {
            return res.status(404).send({
                success: false,
                message: 'User not found'
            });
        }
    });
});

router.get('/device/:deviceId',function(req,res){
    User.findOne({
        userName:req.decoded.userName
    }).populate('devices').exec(function(err,user){
        if(err) throw err;

        var found = false;
        for (device of user.devices) {
            if (device._id == req.params.deviceId) {
                found = true;
                return res.status(200).send({
                    success: true,
                    device: device
                });
            }
        }

        if(!found) {
            return res.status(404).send({
                success: false,
                message: 'Device not found'
            });
        }
    });
});


router.put('/device/:deviceId',function(req,res){
    Device.findOne({
        _id: req.params.deviceId
    }, function(err,device) {
        if (err) throw err;
        if(device){
```

```
            device.name = req.body.name
            device.alarmShouldBeOn = req.body.alarmShouldBeOn
            device.untrigger = req.body.untrigger
            device.save(function(err){
                if (err) throw err;
                return res.status(200).send({
                    success: true,
                    message: 'Device updated',
                    device:  device

                });
            });} else {
            return res.status(404).send({
                success: false,
                message: 'Device does not exist.'
            });
        }
    });
});


router.post('/notificationToken',function(req,res){
    User.findOne({
        userName: req.decoded.userName
    },function(err,user){
        if(err) throw err;
        if(user) {
            user.notificationToken = req.body.notificationToken
            user.save(function(err){
                if(err)    throw err;
                res.json({success:true,
                    message: 'Token Added'})
            });
        }
    });
});


app.use('/api',router);
app.listen(port);
```

APÊNDICE C – CÓDIGO FONTE DO RASTREADOR

```
#include <EEPROM.h>
#include <SoftwareSerial.h>
#include <MPU6050.h>
#include <ArduinoJson.h>
#define DEBUG false
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

typedef enum {active, pairing, triggered, proximity} state;
state deviceState;

MPU6050 accelgyro;
int accSamplesIndex = 0;
int accSamplesBuffer[3][4] = {}; // [0] -> x, [1] -> y, [2] -> z
int ax, ay, az;
int count = 0;

unsigned long lastMillis;
unsigned long enteredActive;
unsigned long timeCounter;

bool alarmTriggered = false;
bool untrigger = false;
bool alarmOn = true;
bool bluetoothConnected = false;
bool sim808Power = false;
bool alreadyHasGpsString = false;
int alreadyPaired = 0;

int sim808PowerKey = 9;
SoftwareSerial bluetoothSerial(5, 6); // RX, TX
SoftwareSerial sim808Serial(7, 8); // RX, TX

String gpsString;

void setup() {

   Serial.begin(19200);
   while (!Serial) {
      ;
   }
   sim808Power = sim808IsOn();
   Serial.println(sim808Power);

   sim808Serial.begin(19200);
   sim808Serial.println("AT"); //recomendacao do fabricante
   delay(500);
   ShowSerialData();

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
   Wire.begin();
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
   Fastwire::setup(400, true);
#endif

   alreadyPaired = EEPROM.read(400);
```

```
    if (alreadyPaired == 1) {
        deviceState = active;
        prepareForActive();

    } else {
        deviceState = pairing;
        prepareForPairing();
    }
}

void loop() {

    switch (deviceState) {

        case active:

            deviceActive();
            if (alarmTriggered && alarmOn){
                deviceState = triggered;
                prepareForTriggered();
            }
            if (bluetoothConnected) {
                deviceState = proximity;
                prepareForProximity();
            }
            if (alreadyPaired == 0) {
                deviceState = pairing;
                prepareForPairing();
            }
            break;

        case pairing:
            devicePairing();
            if (alreadyPaired == 1) {
                leavePairing();
                deviceState = active;
                prepareForActive();
            }
            break;

        case triggered:
            deviceTriggered();
            if (alarmOn == false || untrigger == true) {
                deviceState = active;
                alarmTriggered = false;
                untrigger = false;
                prepareForActive();
            }
            break;

        case proximity:
            deviceProximity();
            if (bluetoothConnected == false) {
                leavePairing();
                deviceState = active;
                prepareForActive();
            }
            break;

    }
```

```
}

void prepareForActive() {
   sim808powerOn();
   gpsString = " ";
   sendStatusAndUpdate();
   gpsPowerOn();
   accelgyro.initialize();
   alreadyHasGpsString = false;
   enteredActive = millis();
   lastMillis = millis();

}

void deviceActive() {
   if (millis() > lastMillis + 1800000) {
      sim808powerOn();
      sendStatusAndUpdate();
      if(alreadyHasGpsString == true){
         sim808powerOff();
      }
      lastMillis = millis();
   }

   if (millis()>enteredActive + 300000 && alreadyHasGpsString == false) {
      sim808powerOff();
      alreadyHasGpsString = true;
      sendStatusAndUpdate();
   } else {
      gpsString = getGpsString();
   }

   sampleAccelerometer();
   alarmTriggered = didMoveSuspiciously();
   bluetoothConnected = isBluetoothConnected();
   delay(300);
}

void prepareForPairing() {
   bluetoothSerial.begin(9600);
}

void devicePairing() {
   char incomingByte = bluetoothSerial.read();
   if (incomingByte == 'h') {
      bluetoothSerial.write("1:5a64f44a4f78f51400");
      bluetoothSerial.write("2:c22a1d");
   }
   if (incomingByte == 'f') alreadyPaired = 1;
}

void leavePairing() {
   EEPROM.write(400,1);
   bluetoothSerial.end();
}

void prepareForProximity() {
   sendStatusAndUpdate();
   sim808powerOff();
   gpsString = "";
```

```
}
void deviceProximity() {
  bluetoothConnected = isBluetoothConnected();
}
void prepareForTriggered () {
  gpsPowerOn();
}
void deviceTriggered() {
  gpsString = getGpsString();
  sendStatusAndUpdate();
}

bool isBluetoothConnected() {
  bluetoothSerial.begin(9600);
  bluetoothSerial.print("5a64f44a4f78f51400");
  bluetoothSerial.setTimeout(3000);
  bool pong = bluetoothSerial.find("pong");
  bluetoothSerial.end();
  return pong;
}

void sampleAccelerometer() {
  accelgyro.getAcceleration(&ax, &ay, &az);
  accSamplesBuffer[0][accSamplesIndex] = ax;
  accSamplesBuffer[1][accSamplesIndex] = ay;
  accSamplesBuffer[2][accSamplesIndex] = az;

  accSamplesIndex++;

  if (accSamplesIndex == 4) {
    accSamplesIndex = 0;
  }
}

bool didMoveSuspiciously() {
  if (millis() < 15000 + enteredActive) {
    return false;   //Alarme nÃ£o funciona nos primeiros dez segundos para esperar o acelerÃ´metro estabilizar
  }

  long int sumAccSampleX = 0;       //Soma das amostras do acelerÃ´metro no eixo x
  long int sumAccSampleY = 0;       //Soma das amostras do acelerÃ´metro no eixo y
  long int sumAccSampleZ = 0;       //Soma das amostras do acelerÃ´metro no eixo z
  long int sumAverageX = 0;         //MÃ©dia das amostras do acelerÃ´metro no eixo x
  long int sumAverageY = 0;         //MÃ©dia das amostras do acelerÃ´metro no eixo y
  long int sumAverageZ = 0;     //MÃ©dia das amostras do acelerÃ´metro no eixo z


  for (int i = 0; i <= 3; i++) {
    if (i != accSamplesIndex) {
      sumAccSampleX = accSamplesBuffer[0][i] + sumAccSampleX;
      sumAccSampleY = accSamplesBuffer[1][i] + sumAccSampleY;
      sumAccSampleZ = accSamplesBuffer[2][i] + sumAccSampleZ;
    }
  }

  sumAverageX = sumAccSampleX / 3;
  sumAverageY = sumAccSampleY / 3;
  sumAverageZ = sumAccSampleZ / 3;

  if     (abs(sumAverageX - accSamplesBuffer[0][accSamplesIndex]) > 600) return true;
```

```
    else if (abs(sumAverageY - accSamplesBuffer[1][accSamplesIndex]) > 600) return true;
    else if (abs(sumAverageZ - accSamplesBuffer[2][accSamplesIndex]) > 600) return true;

    return false;
}

String getGpsString() {

    String fullGpsString = "";
    int countCommas = 0;
    char incomingByte;
    int firstCommaIndex=0;
    int secondCommaIndex=0;

    sim808Serial.begin(19200);
    delay(200);
    sim808Serial.println("AT+CGNSINF");

    while(sim808Serial.available() == 0) {
    }
    while (sim808Serial.available() > 0) {
        incomingByte = sim808Serial.read();
        fullGpsString.concat(incomingByte);
    }


    for(int i=0;i<fullGpsString.length();i++){
        if (fullGpsString[i] == ','){
            countCommas++;
        }
        if (countCommas == 2) {
            firstCommaIndex = i+2;
        }
        if (countCommas == 4) {
            secondCommaIndex = i+1;
        }
    }

    Serial.println(fullGpsString.substring(firstCommaIndex,secondCommaIndex));
    return fullGpsString.substring(firstCommaIndex,secondCommaIndex);
}

void sendStatusAndUpdate() {

    sim808powerOn(); //Garantir que o sim808 estÃ¡ ligado
    bluetoothSerial.end();
    sim808Serial.begin(19200);
    delay(100);
    sim808Serial.listen();
    sim808Serial.println("AT+CSQ");
    delay(100);
    ShowSerialData();
    sim808Serial.println("AT+CGATT?");
    delay(100);
    ShowSerialData();
    sim808Serial.println("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"");//setting the SAPBR, the connection type is using gprs
    delay(1000);
    ShowSerialData();
    sim808Serial.println("AT+SAPBR=3,1,\"APN\",\"CMNET\"");//setting the APN, the second need you fill in your local apn server
    delay(500);
```

```
    ShowSerialData();
    sim808Serial.println("AT+SAPBR=1,1");//setting the SAPBR, for detail you can refer to the AT command mamual
    delay(1850);
    ShowSerialData();
    sim808Serial.println("AT+HTTPINIT"); //init the HTTP request
    delay(200);
    ShowSerialData();
    sim808Serial.print("AT+HTTPPARA=\"URL\",\"guard-app.herokuapp.com/deviceStatus?deviceId=5a64f44a4f78f51400c22a1d");//
setting the httppara, the second parameter is the website you want to access

    if (alarmOn == true) {
        sim808Serial.print("&alarmOn=true");
    } else {
        sim808Serial.print("&alarmOn=false");
    }

    sim808Serial.print("&gpsString=");
    if(deviceState == proximity) {
        sim808Serial.print("nearby");
    } else {
        sim808Serial.print(gpsString);
    }

    sim808Serial.print("&state=");
    sim808Serial.print(String(deviceState));
    sim808Serial.print("\"");
    sim808Serial.println("");

    delay(200);
    ShowSerialData();
    sim808Serial.println("AT+HTTPACTION=0");//submit the request
    delay(10000);//the delay is very important, the delay time is base on the return from the website, if the return datas are very large,
the time required longer.
    ShowSerialData();
    sim808Serial.println("AT+HTTPREAD");// read the data from the website you access
    delay(300);


    sim808Serial.find("\r");
    sim808Serial.find("\r");
    sim808Serial.find("\r");

    String json = sim808Serial.readString();
    StaticJsonBuffer<200> jsonBuffer;
    JsonObject& root = jsonBuffer.parseObject(json);
    if (root.success()) {
        //bool success = root["success"];
        alarmOn = root["a"];
        alreadyPaired = root["p"];
        untrigger = root ["u"];
        if(alreadyPaired == 0) {
            EEPROM.write(400,0);
        }
    } else {
        Serial.println("parsing failed");
        Serial.println("O Json eh");
        Serial.println(json);
    }

    sim808Serial.end();
```

```
}

void ShowSerialData()
{
  while (sim808Serial.available() != 0)
    Serial.write(sim808Serial.read());
}


bool sim808IsOn() {
  sim808Serial.begin(19200);
  delay(100);
  sim808Serial.setTimeout(3000);
  sim808Serial.println("AT");
  delay(500);
  if(sim808Serial.find("OK")){
    return true;
  } else {
    return false;
  }
  sim808Serial.setTimeout(1000);
}

void sim808powerOn() {

  if(sim808Power == false){
    digitalWrite(sim808PowerKey, LOW);
    delay(1000);
    digitalWrite(sim808PowerKey, HIGH);
    delay(15000);
  }
  sim808Serial.setTimeout(1000);
  sim808Power = true;
}

void sim808powerOff() {
  if(sim808Power == true) {
    digitalWrite(sim808PowerKey, LOW);
    delay(1000);
    digitalWrite(sim808PowerKey, HIGH);
    delay(2000);
    digitalWrite(sim808PowerKey, LOW);
  }
  sim808Power = false;
}

void gpsPowerOn() {
  sim808powerOn(); //Garantir que o sim808 estÃ¡ ligado
  sim808Serial.begin(19200);
  delay(100);
  sim808Serial.println("AT+CGNSPWR=1");
  delay(100);
  ShowSerialData();
  delay(100);

}

void gpsPowerOff() {
  sim808Serial.write("AT+CGNSPWR=0");
```

```
    delay(100);
}
```