UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

GABRIEL CÉSAR ALVES LIMA

EXTENSÃO DA API *PREDETECT*: UTILIZANDO O *BLUETOOTH LOW ENERGY* PARA DETECÇÃO DE PRESENÇA DE DISPOSITIVOS MÓVEIS EM AMBIENTES *INDOOR*

QUIXADÁ

2018

GABRIEL CÉSAR ALVES LIMA

EXTENSÃO DA API *PREDETECT*: UTILIZANDO O *BLUETOOTH LOW ENERGY* PARA DETECÇÃO DE PRESENÇA DE DISPOSITIVOS MÓVEIS EM AMBIENTES *INDOOR*

Monografia apresentada no curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação. Área de concentração: Computação.

Orientador: Prof. Dr. Jefferson de Carvalho Silva

QUIXADÁ

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

L698e Lima, Gabriel Cesar Alves.
Extensão da API PreDetect : utilizando o Bluetooth Low Energy para detecção de presença de dispositivos móveis em ambientes indoor / Gabriel Cesar Alves Lima. – 2018.
57 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2018.
Orientação: Prof. Dr. Jefferson de Carvalho Silva.

1. Interface de programação de aplicações. 2. Android (Programa de computador). 3. Bluetooth de baixa energia. 4. Sistema de localização indoor. I. Título.

CDD 005

GABRIEL CÉSAR ALVES LIMA

EXTENSÃO DA API *PREDETECT*: UTILIZANDO O *BLUETOOTH LOW ENERGY* PARA DETECÇÃO DE PRESENÇA DE DISPOSITIVOS MÓVEIS EM AMBIENTES *INDOOR*

Monografia apresentada no curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação. Área de concentração: Computação.

Aprovada em: ____/____/____.

BANCA EXAMINADORA

Prof. Dr. Jefferson de Carvalho Silva (Orientador)
Campus Quixadá
Universidade Federal do Ceará – UFC

Prof. Me. Luis Rodolfo Rebouças Coutinho
Campus Quixadá
Universidade Federal do Ceará - UFC

Prof. Me. Wagner Guimarães Al-Alam
Campus Quixadá
Universidade Federal do Ceará - UFC

Aos meus pais, Francisco César Sabino Lima e Marinete Alves Coelho.

AGRADECIMENTOS

Agradeço aos meus pais que sempre me incentivaram e investiram no meu futuro.
Agradeço ao meu orientador, Professor Doutor Jefferson de Carvalho Silva, pela paciência, disponibilidade e por me orientar durante o projeto.
Agradeço aos professores Luis Rodolfo Rebouças Coutinho e Wagner Guimarães Al-Alam, pela disponibilidade em participar da banca desse trabalho e pelas colaborações e sugestões.
Aos meus amigos e colegas de graduação pela parceria durantes esses 4 anos.
A todos que direta e indiretamente fizeram parte da minha graduação.

“Você nunca alcança o sucesso verdadeiro a menos que você goste do que está fazendo.”

(Dale Carnegie)

RESUMO

Os sistemas de posicionamento desempenham um papel cada vez mais nítido na vida das pessoas, já que as mesmas permanecem maior parte do tempo em ambientes fechados (*indoor*). Nesse contexto, surgem um gama de possibilidades, como fornecimento de conteúdo acionado por localização, publicidade baseada na localização, rastreamento de ativos, entre outros. Mas, diferente do posicionamento externo, o posicionamento *indoor* não possui uma tecnologia consolidada como o GPS (*Global Positioning System*). Além disso, os sinais GPS não apresentam boa performance em ambientes fechados. Sendo assim, foram desenvolvidas alternativas para o posicionamento *indoor*. A abordada neste trabalho, utiliza a tecnologia *Bluetooth*. Redes Bluetooth possuem algumas vantagens, como baixo custo e além de uma grande variedade de dispositivos compatíveis. O foco deste documento é adicionar uma nova possibilidade de localização à API PreDetect, originalmente projetada para trabalhar apenas com sinal WiFi.

Palavras-chave: API. Android. *Bluetooth Low Energy*. Localização *indoor*.

ABSTRACT

Positioning systems play an increasingly clear role in people's lives, as they remain most of the time indoors. In this context, a range of possibilities arise, such as locally driven content delivery, location-based advertising, asset tracking, and more. But, unlike external positioning, indoor positioning does not have a consolidated technology like GPS (Global Positioning System). In addition, GPS signals do not perform well indoors. Therefore, alternatives have been developed for indoor positioning. The approach used in this work uses Bluetooth technology. Bluetooth networks have some advantages, such as low cost and a wide variety of compatible devices. The focus of this document is to add a new location possibility to the PreDetect API, originally designed to work with WiFi signal only.

Keywords: API. Android. Bluetooth Low Energy. Indoor Positioning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ART	Android Runtime
ARMA	Autoregressive Moving Average
BLE	Bluetooth Low Energy
CPU	Central Process Unit
CSV	Comma Separated Values
CSS	Cascading Style Sheets
DEX	Dalvik Executable
GPS	Global Positioning System
GPU	Graphics Processing Unit
GPM	Global Positioning Module
HTML	Hypertext Markup Language
JSON	Javascript Object Notation
KNN	K-Nearest Neighbor
MB	Megabyte
MVC	Model View Controller
MAC	Medium Access Control
NoSQL	Not Only SQL
NBC	Naive Bayes Classifier
RSSI	Received Signal Strength Indicator
RAM	Random Access Memory
SIG	Special Interest Group

SSID	Service Set Identifier
TDOA	Time Difference Of Arrival
UFC	Universidade Federal do Ceará
WiFi	Wireless Fidelity

SUMÁRIO

1 INTRODUÇÃO

Com o avanço da tecnologia, os computadores passaram a ser cada vez menores, mais potentes e acessíveis, mudando consideravelmente a forma como são utilizados, permitindo ao seus usuários terem acesso a informações independente de suas localizações.

Um dos mais comuns exemplos de tecnologia é o *smartphone*. Um *smartphone* fornece uma grande variedade de informações através dos seus sensores, como a sua velocidade, temperatura ambiente, orientação e até mesmo a localização geográfica e direção.

Considerando que as pessoas passam cerca de 80% a 90% do seu tempo em ambientes fechados (*indoor*) (SIMÕES, 2015), surgiu uma grande variedade de serviços que utilizam a localização como principal funcionalidade. Alguns serviços disponibilizam informações sobre produtos para os clientes a partir da proximidade de determinada prateleira; outros exibem o menor caminho em um ambiente *indoor*, ou mesmo, auxiliam a mobilidade dos visitantes de um museu até uma obra desejada, por exemplo.

Um meio de oferecer esses serviços é utilizando a rede WiFi, por estar presente na maioria dos dispositivos móveis. Contudo, existem algumas limitações, principalmente no que se refere a sua precisão e eficiência energética. Para contornar essas limitações pode ser utilizado o *Bluetooth Low Energy* (BLE) para localização *indoor*, devido ao seu baixo custo e baixo consumo de energia se comparado ao WiFi, e também por estar presente em diversos dispositivos, desde sensores de batimento cardíaco a *smartphones*.

Visando a problemática apresentada, esse trabalho foca na extensão da API *PreDetect*, adicionando uma nova funcionalidade capaz de detectar a presença de dispositivos móveis utilizando a rede *Bluetooth* na API *PreDetect*, disponibilizada para a plataforma *Android* desenvolvida por Filho (2016), aluno da Universidade Federal do Ceará no Campus de Quixadá . Assim como proposto na API *PreDetect*, também se utilizará dos algoritmos de posicionamento baseados no indicador de força do sinal recebido (RSSI) para detecção de presença de dispositivos móveis.

A API *PreDetect* é uma API para a plataforma *Android* com o objetivo de detectar dispositivos móveis em ambientes *indoor*. A API em sua antiga versão permite detectar dispositivos móveis apenas de uma forma, através da rede WiFi. Para facilitar a leitura será utilizada a nomenclatura PreDetect 2.0 para a versão desenvolvida nesse trabalho.

A API disponibiliza duas funcionalidades principais, listar as informações (MAC, SSID, RSSI, Distância aproximada) de todas as redes WiFi, próximas ao dispositivo e a

possibilidade de se obter a porcentagem de tempo que um dispositivo ficou próximo a determinada rede.

A API *PreDetect* foi projetada com uma arquitetura monolítica, sem modularidade interna, mantendo uma aplicação com alto acoplamento. Na Figura 1 é possível visualizar essa estrutura.

Figura 1 – Arquitetura da API *PreDetect*

Fonte – Elaborado pelo autor.

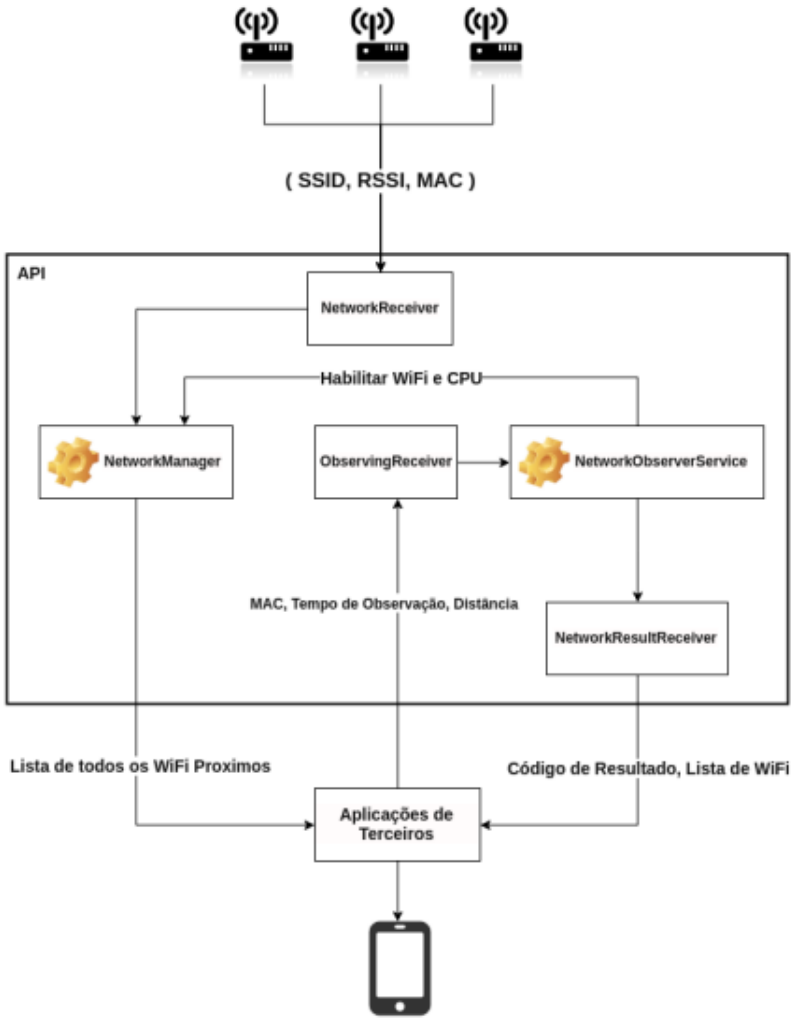
A API *PreDetect* utiliza a API de *Network* da plataforma *Android*, para obter os dados das redes WiFi próximas a determinado dispositivo.

Também é utilizada a API *WifiManager*, fornecida pela a plataforma *Android*, para obter o MAC, SSID, RSSI da redes WiFi. Com os dados adquiridos é enviada uma notificação via *broadcast*, e as classes que desejam receber a notificação seriam configuradas previamente.

Existem duas formas de requisitar esse serviço. Uma delas é implementando a interface *WiFiObserver* para então receber uma instância da classe *NetworkManager* e após registrar-se os dados seriam enviados diretamente para o método definido pela interface. A outra forma, seria requisitar esse serviço por meio de uma *Intent* (intenção), nesse caso, terá de ser criado um *broadcast receiver*, responsável por receber mensagens quando um evento especifico é executado.

Toda a execução da API *PreDetect*, ocorre em segundo plano utilizando a classe *Service* fornecida pela a plataforma *Android*. Um ponto a ser ressaltado é o consumo de energia quando se utiliza serviços em segundo plano, para evitar o consumo indevido de bateria, quando o serviço é finalizado, ambos os *WakeLocks* do processador e do WiFi, mecanismos de gerenciamento de bateria que desabilita os componentes que não estão sendo utilizados, são liberados.

Figura 2 – Visão geral do funcionamento da API *PreDetect*



Fonte – Filho (2016).

Tanto a documentação quanto o código fonte da API foram disponibilizados na plataforma *Github*[1], que pode ser acessado em <https://github.com/dielfilho/predetect>. Na documentação é descrita a forma de utilização, assim como, as classes que serão necessárias implementar para realizar a atividade desejada.

Para disponibilizar a API pronta para utilização foi necessário outra ferramenta para executar a compilação do código fonte diretamente do repositório da plataforma *Github*. A ferramenta utilizada foi o *JitPack*[2].

Nesse contexto, o objetivo deste trabalho consiste em remodelar a API *PreDetect* para para uma arquitetura modular, desacoplando-a em módulos baseados no tipo de detecção e também implementar um novo módulo que forneça um meio de detectar a localização relativa de

[1] https://github.com
[2] https://jitpack.io

um dispositivo móvel em um ambiente *indoor* utilizando a rede do *Bluetooth Low Energy* (BLE).

Também é almejado a contribuição para a comunidade de desenvolvedores na criação de aplicações móveis para ambientes *indoor*, atentando-se as limitações da plataforma *Android* que tem um limite de métodos que podem ser compilados pelo sistema, produzindo uma biblioteca onde pode-se escolher com qual modo de detecção deseja trabalhar, eliminando código desnecessário na hora de criar o aplicativo. Além dessa contribuição, também será produzida a documentação completa *online* para ao módulo proposto para que terceiros possam criar outras aplicações sobre a API proposta e, até mesmo, adicionar novas funcionalidades.

Para validação do módulo, será considerado a precisão no cálculo da distância, eficiência ao utilizar os recursos e a eficácia ao realizar checagens. Também será analisado quais ambientes são mais adequados para a utilização de cada módulo.

Os próximos capítulos estão organizados da seguinte maneira: no Capítulo 2 serão apresentados os conceitos técnicos e teóricos necessários para a realização do trabalho; no Capítulo 3 serão abordados os trabalhos relacionados; no Capítulo 4 serão apresentados os procedimentos metodológicos; o Capítulo 5 descreverá os resultados e discussões; e o Capítulo 6 apresenta as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo são apresentados os principais conceitos utilizados para entendimento e execução deste trabalho. Na Seção 2.1 é apresentado o *Bluetooth Low Energy* e como ele será utilizado para emitir um sinal em *broadcast* para que *smartphones* com sistema operacional *Android*, discutido na seção 2.2, possam calcular as distâncias baseando-se no RSSI, com a formula apresentada na Seção 2.3, e para reduzir ruídos foi aplicado o *Kalman Filter*, descrito na Seção 2.4.

2.1 *Bluetooth Low Energy*

O *Bluetooth Low Energy (BLE)* ou *Bluetooth* de Baixa Potência é um tipo de tecnologia de conexão sem fio desenvolvida pelo *Bluetooth Special Interest Group* (*Bluetooth* SIG) e pode ser entendida como uma especificação, ou funcionalidade que integra o *Bluetooth* desde 2010, a partir de sua versão 4.0 (BLUETOOTH SPECIAL INTEREST GROUP, 2010).

O *Bluetooth Low Energy*, como o próprio nome sugere, surgiu com o objetivo de ser mais eficiente que o *Bluetooth* Clássico no comportamento energético. É importante destacar que o BLE é projetado para operações de baixíssima potência e otimizado para transferência dados em pequena quantidade. Outro ponto a ser ressaltado é que o *Bluetooth Low Energy* possui *micro-location*, possibilitando obter a localização com uma acurácia abaixo de 1 metro. No Quadro 1, é exibido um comparativo entre as tecnologias *Bluetooth*.

Quadro 1 – Comparativo entre *Bluetooth Low Energy* e *Bluetooth Clássico*

Especificações Técnicas	*Bluetooth Low Energy*	*Bluetooth Clássico*
Otimizado para...	Pequena quantidade de dados	*Streaming* contínuo de dados
Frequência de banda	2.4 GHz (2.402 GHz a 2.480 GHz)	2.4 GHz (2.402 GHz a 2.480 GHz)
Canais	40 canais (2 MHz)	79 canais (1 MHz)
Consumo de energia	0.01 até 0.5 (de acordo com o uso)	1 (valor de referencia)
Taxa de dados	1 Mb/s	1 - 3 Mb/s
Modulação	GFSK	GFSK
Escravos ativos	Não definido, depende da aplicação	7 escravos
Segurança	128-bit AES	56 até 128-bit
Latência	6ms	100 ms
Topologias de Rede	Ponto-a-Ponto (incluindo piconet) Broadcast Mesh	Ponto-a-Ponto (incluindo piconet)

Fonte – Adaptado de Bluetooth Special Interest Group (2018) e Laird (2013)

No Quadro 1, é salientado que o BLE possui várias melhorias em relação ao *Bluetooth* Clássico. Podemos destacar principalmente a baixa latência permitindo melhor qualidade de conexão. Também há de se destacar a variedade de topologias permitindo seu uso nos mais diversos âmbitos, além de oferecer opções de segurança até o nível governamental.

Neste trabalho, o BLE foi utilizado como um ponto de referência. O BLE emitirá o sinal que será detectado pelo módulo BLE e possibilitará a obtenção da localização relativa do *smartphone*, baseada na transformação do RSSI em unidades de distância.

2.1.1 Beacon BLE

Um *beacon BLE* é um dispositivo que emite um sinal intermitente de ondas de rádio através do *Bluetooth low Energy* (BLE) em *broadcast* para todos os dispositivos que estejam em um raio de cerca de 100 metros (ZAFARI; PAPAPANAGIOTOU,).

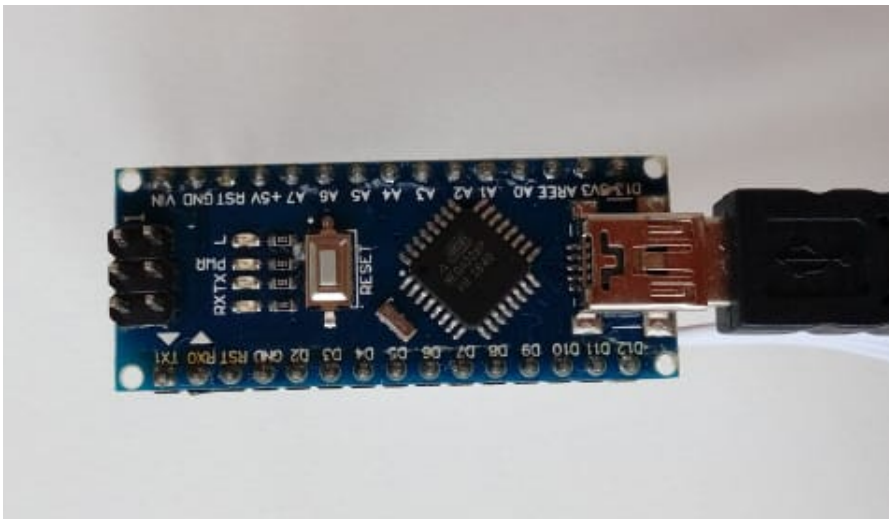
Os autores Zafari e Papapanagiotou () também descrevem os *Beacons* como ferramentas para identificar objetos inteligentes próximos e mostrar conteúdos específicos de acordo com sua localização e proximidade aos *Beacons*.

No entanto, neste trabalho o foco será em dispositivos móveis que utilizam o Sistema Operacional *Android* e eles que serão responsáveis por identificar os *Beacons*.

Na Figura 3 é possível visualizar o Arduino Nano que será utilizado como *Beacon* e

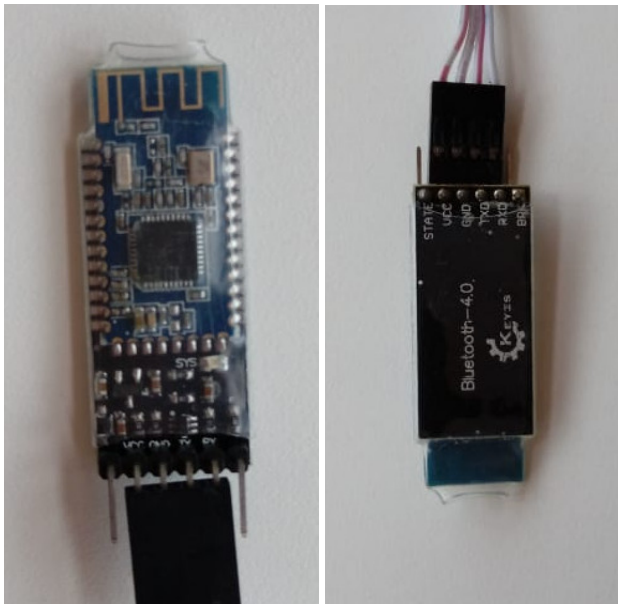
na Figura 4 o módulo BLE quer será responsável por emitir o sinal em *broadcast*, para que seja encontrado pelos dispositivos.

Figura 3 – Arduino Nano V3 ATmega328P CH340g



Fonte – Elaborado pelo autor.

Figura 4 – Módulo HM-10 *Bluetooth Low Energy*
Keyes



Fonte – Elaborado pelo autor.

2.2 *Android*

Desenvolvido pelo *Google*, *Android* é uma plataforma de desenvolvimento de aplicações móveis que possui o sistema operacional baseado no *kernel* 2.6 do *Linux*, responsável por gerenciar toda a memória, processos, *threads*, segurança de arquivos, pastas, além de gerenciamento de redes e *drivers* (LECHETA, 2013). Atualmente, houve uma mudança na linguagem utilizada para o desenvolvimento, migraram do Java para o *Kotlin*, uma linguagem de programação de tipagem estática para aplicações modernas multiplataforma.

A Figura 5 apresenta uma visão geral da pilha de software do *Android* e suas divisões.

Figura 5 – A pilha de software do *Android*.

Fonte – Adaptada de Google (2018)

No topo da pilha se encontra a camada dos Aplicativos do Sistema, responsável por encapsular todas as aplicações básicas, como agenda de contatos, navegador de internet, entre outras. É nessa camada que estão as aplicações desenvolvidas por terceiros (SIMÕES, 2015).

Logo abaixo, na camada *Java API Framework* há o gerenciamento dos recursos entre programa e processos.

Na camada inferior está as Bibliotecas Nativas, que oferecem suporte a operações com vídeo e áudio, entre outras. No mesmo nível se localiza a camada *Android Runtime* (ART). O ART é projetado para executar várias máquinas virtuais em dispositivos de baixa memória executando arquivos DEX, um formato de *bytecode* projetado especialmente para *Android*, otimizado para oferecer consumo mínimo de memória (GOOGLE, 2018).

Posteriormente temos a Camada de Abstração do Hardware. Essa camada consiste em módulos de biblioteca, que implementam uma interface para um tipo específico de

componente de hardware, como o módulo *bluetooth* (GOOGLE, 2018).

A seguir temos a camada *Kernel Linux*, responsável por todo o gerenciamento de memória, *Bluetooth*, *WiFi* e energia. Visando uma melhor performance as bibliotecas de GPU (*Graphics Processing Unit*, ou Unidade de Processamento Gráfico), e CPU (*Central Process Unit*, ou Unidade Central de Processamento) foram compiladas para código nativo, consumindo menos memória e pouca bateria (BRAHLER, 2010).

E por fim, a camada de Gerenciamento de Energia. Para prolongar a duração da bateria, o *Android* adiciona continuamente novos recursos e otimizações para ajudar a plataforma a otimizar o comportamento dos aplicativos (GOOGLE, 2018).

O módulo desenvolvido nesse trabalho foi elaborado para essa plataforma, utilizando de todas as funcionalidades providas pela mesma. O módulo BLE, opera na camada de Aplicativos do Sistema. Devido a recente mudança na linguagem padrão de desenvolvimento, pode existir incompatibilidades entre as funcionalidades existentes e o ambiente atual de desenvolvimento, com isso em mente, foram realizadas correções nas funcionalidades existentes, para que estejam em completo funcionamento.

2.2.1 Bluetooth Low Energy Platform

O suporte nativo para a plataforma *Bluetooth Low Energy* (BLE), foi introduzido no *Android* 4.3 *KitKat* (API *level* 18), permitindo que os aplicativos possam descobrir dispositivos, consultar serviços e transmitir informações (GOOGLE, 2018).

Entretanto, existem algumas problemáticas no BLE do *Android* 4.3 *KitKat*, como recursos ausentes, problemas de confiabilidade, consumo de energia da bateria e também por não ser *Thread Safe*. Felizmente, no *Android* 5.0 *Lollipop* (API *level* 21), traz correções para os principais problemas e adiciona suporte para muitos dos recursos ausentes, como periféricos BLE, varredura aprimorada e menor consumo de energia (ARGENOX TECHNOLOGIES, 2018).

O BLE na plataforma *Android* é um protocolo assíncrono baseado no envio de mensagens entre dois dispositivos, todo o resultado deve ser capturado através de *callbacks* para os vários tipos de mensagens e eventos que podem acontecer. Como vários aplicativos podem se comunicar por meio do BLE simultaneamente, o sistema usa um componente central para toda a comunicação do BLE com a qual seu aplicativo se comunica. Outra característica é que a descoberta de dispositivos e a comunicação são realizadas de forma separada (ERIK HELLMAN, 2018).

2.3 RSSI-*Based*

Na técnica de posicionamento RSSI-*Based* a distância entre o emissor e o receptor é calculada tendo como base a força do sinal recebido (RSSI). A distância pode ser calculada através da relação entre a potência do sinal e sua taxa de perda em função da distância a qual pode ser obtida pelo cálculo da Perda no Espaço Livre (*Free Space Path Loss*).

Adequando a fórmula para calcular o RSSI apresentada por KUROSE e Ross (2006), para estimação da distância, temos:

$$d = 10^{\frac{R-A}{-10.n}} \tag{2.1}$$

onde d é a distância estimada, R é a intensidade do sinal que chega. A é a intensidade do sinal medido a 1 metro do emissor ou ponto de acesso. A variável n representa a perda da intensidade do sinal proveniente do emissor. No espaço livre n é igual a 2 e para caminhos com obstruções, n tem um valor entre 4 e 5 (KUROSE; ROSS, 2006).

Essa técnica será utilizada no módulo para estimar a distância entre o *Beacon BLE* e o *smartphone*. Para compensar a variedade de sinais existentes nos locais, possíveis interferências e também pelo modelo de *Beacon BLE* adotado serão utilizados os seguintes valores: a intensidade do sinal medido a 1 metro do emissor será uma constante com valor -62; a perda de sinal proveniente do emissor será igual a 3.5.

2.4 *Kalman Filter*

Os valores de RSSI são fortemente influenciados pelo ambiente e, consequentemente, altos níveis de ruído. Esse ruído é, por exemplo, causado por reflexos de múltiplos caminhos: os sinais refletem contra objetos no ambiente, como paredes e móveis (BULTEN; ROSSUM; HASELAGER, 2016).

Segundo Mindell e Moss (2018) é necessário filtrar os ruídos das medições e fazer a melhor estimativa possível ao mesmo tempo que precisa saber o quanto as boas ou ruins são as estimativas, em um sentido estatístico, para poder evitar falsos positivos e o mais indicado para essa tarefa é o *Kalman Filter*.

O *Kalman Filter*, tem como objetivo minimizar o erro quadrático médio entre os dados reais e estimados. Assim, fornece a melhor estimativa dos dados no sentido do erro

quadrático médio (THACKER; LACEY, 1998).

Na Equação 2.2, é mostrada de forma simplificada a formula do *Kalman Filter*.

$$\hat{X_k} = K_k.Z_k + (1 - K_k).X_{k-1} \tag{2.2}$$

Onde $\hat{X_k}$ é o valor estimado atualmente, K_k o *Kalman Gain*, Z_k o valor medido e X_{k-1} o valor estimado anteriormente. Nesse contexto o *Kalman Gain* é o unico valor desconhecido, pois ele é calculado a cada iteração.

O *Kalman Gain* pode ser calculado da seguinte forma:

$$K_k = L_{k-1}.(I - K_{k-1})/(L_{k-1} + V) \tag{2.3}$$

Onde L_k é a erro de covariância do valor anterior, V é a *measurement noise*, I é o valor atual e K_{k-1} é valor do *Kalman Gain* anterior.

3 TRABALHOS RELACIONADOS

Neste Capítulo serão apresentados alguns trabalhos relacionados destacando as semelhanças e diferenças com o desenvolvido neste trabalho. Na Seção 3.1 é apresentada a biblioteca em que o trabalho deste documento visa aperfeiçoar.

3.1 PreDetect

Em Filho (2016), é apresentada uma API para detecção de presença utilizando o *WiFi* e uma aplicação para a plataforma *Android*, com o objetivo de realizar a checagem de presenças dos estagiários dentro das salas localizadas na UFC - Campus Quixadá. A partir da localização do *smartphone*, se estivesse próximo a rede *WiFi* do ambiente de estágio e no horário cadastrado sua presença será realizada automaticamente.

A API desenvolvida por Filho (2016), utiliza a API de *Network* da plataforma *Android*, para que seja possível obter os dados das redes próximas a determinado dispositivo. Também é utilizada a API *WifiManager*, fornecida pela plataforma *Android*, para obter os dados das redes, como MAC, SSID, RSSI, entre outros. Com esses dados a API já consegue informar a distância do dispositivo móvel até o ponto de referência, o *WiFi*. Outra funcionalidade fornecida pela API é de checar se um dispositivo móvel está presente dentro de um raio de distância por um determinado tempo.

A aplicação desenvolvida por Filho (2016) utiliza a arquitetura Cliente/Servidor. Na parte do servidor fica armazenado o *Media Access Control* (MAC) do ponto de acesso da sala junto aos dias, horários das atividades, horários de checagem de presença e porcentagem mínima para validar a presença.

Na parte do cliente, caso não possua conexão com um ponto de acesso a internet os dados seriam comprometidos por não serem armazenados localmente no dispositivo.

Assim como Filho (2016), nosso trabalho foca em localizar *smartphones* em ambientes *indoor* utilizando redes sem fio. Porém seu trabalho consiste apenas no uso de redes *WiFi* para detecção de presença e no caso desse documento fornecemos a possibilidade do uso do *Bluetooth Low Energy*.

3.2 *Bluetooth* GPM

No trabalho de Bekkelien, Deriaz e Marchand-Maillet (2012) é desenvolvido um sistema de localização *indoor* com o objetivo de integrá-lo ao *Global Positioning Module* (GPM), um *framework* que combina vários módulos de posicionamento *indoor* e *outdoor* em dispositivos móveis, desenvolvido no *Institute of Services Science* da Universidade de Genebra.

O sistema *Bluetooth* GPM foi desenvolvido para a plataforma *Android*. O sistema utiliza a técnica de análise de cena, essa técnica consiste no mapeamento da potência do sinal recebido (RSSI) pelo receptor em locais já conhecidos, com duas fases, uma fase *offline* onde seriam coletadas as medições do local e outra fase *online* onde seria comparado os indicadores de força de sinal recebido (RSSI) dos dispositivos *Bluetooth* do local com as medições realizadas na fase *offline* utilizando os algoritmos *K-Nearest Neighbor* (KNN) e *Naive Bayes Classifier* (NBC), a fim de determinar o posicionamento do dispositivo. A precisão obtida foi de aproximadamente 1,5 metros.

O sistema *Bluetooth* GPM se assemelha bastante ao proposto no documento, no que diz respeita a criação de módulo para um sistema já existente, até mesmo na utilização da plataforma *Android*, mas se diferencia ao utilizar o *Bluetooth* enquanto o *PreDetect 2.0* utiliza o *Bluetooth Low Energy*. Outro ponto diferente, é que a API *PreDetect 2.0* permite visualizar a percentagem de presença em um determinado raio, enquanto o *Bluetooth* GPM visa mostrar somente a localização de um determinado dispositivo.

3.3 *BLE Indoor Positioning*

BLE Indoor Positioning é uma biblioteca *Android* que é capaz de estimar locais com base em pacotes de publicidade recebidos de *Beacons BLE*. Ela foi desenvolvida pela empresa NeXenio (2018) e é um projeto *open source*[1] e está disponível no seguinte endereço <https://github.com/neXenio/BLE-Indoor-Positioning>.

Para indicar a localização do dispositivo a biblioteca *BLE Indoor Positioning* utiliza a técnica de Multilateração usando *Beacons BLE*. A técnica funciona da seguinte maneira, são colocados vários *Beacons BLE* em locais diferentes que recebem o mesmo sinal do dispositivo, e uma unidade central de processamento é responsável por calcular a posição através da medição

[1] O termo "open source"refere-se a algo que as pessoas podem modificar e compartilhar porque seu design é acessível publicamente.

da diferença do tempo da chegada (TDOA, *Time Difference Of Arrival*) do sinal dos diferentes *beacons*.

Como o RSSI possuí bastante ruído, a biblioteca permite a utilização de alguns filtros para estimar com mais precisão a localização. Os filtros disponíveis, são: o filtro por média móvel; o filtro por média móvel autorregressiva (ARMA) e o *Kalman Filter*.

Assim, como a biblioteca *BLE Indoor Positioning* o trabalho desenvolvido também utiliza filtros para melhor estimativa da localização, o filtro utilizado o *Kalman Filter*. A biblioteca *BLE Indoor Positioning* também permite observar as atualizações de certos *Beacons BLE*, em contrapartida não permite adicionar nenhuma tipo de regra adicional, como um raio especifico de observação.

3.4 *iOS-Indoor-SDK*

iOS-Indoor-SDK é uma biblioteca *iOS* que permite mapeamento baseado em *beacons* BLE em tempo real e localização interna. Ela foi desenvolvida pela empresa Estimote (2018) e é um projeto *open source* e está disponível no seguinte endereço <https://github.com/Estimote/iOS-Indoor-SDK>.

A biblioteca permite a criar novas experiências móveis, desde análises no local de eventos e marketing de proximidade até pagamentos sem contato e compras personalizadas. Porem, a biblioteca trabalha exclusivamente com os *Beacons BLE* da empresa *Estimote*.

Essa biblioteca oferece uma gama de possibilidades e assim como a API *PreDetect 2.0* possui opções de observar determinado dispositivos em segundo plano, também possibilita limitar o espaço a ser utilizado mas em contrapartida não permite limitar o intervalo de tempo a ser observado. Outra diferença é que ela só funciona para *Beacons BLE* específicos enquanto o proposto não tem essa limitação.

O Quadro 2 mostra as semelhanças e diferenças dos trabalhos citados neste capítulo com o trabalho proposto, em relação às suas principais características.

Quadro 2 – Comparativo entre os trabalhos relacionados e o trabalho proposto.

	PreDetect	*Bluetooth GPM*	*BLE Indoor Positioning*	*iOS-Indoor-SDK*	*PreDetect 2.0*
Utiliza o *Bluetooth Low Energy*	Não	Não	Sim	Sim	Sim
Utiliza a técnica RSSI-*Based*	Sim	Não	Sim	Não	Sim
Funciona em qualquer modelo de *Beacon BLE*	Não	Não	Sim	Não	Sim
Permite observar dispositivos por um determinado intervalo de tempo	Sim	Não	Não	Não	Sim
Permite observar dispositivos baseado na distância	Sim	Não	Sim	Sim	Sim
Utiliza algum filtro ou técnica para melhorar a estimativa da distância	Não	Sim	Sim	Sim	Sim
Possibilita adicionar novas tecnologias de detecção	Não	Sim	Não	Sim	Sim

Fonte – Elaborada pelo autor.

4 PROCEDIMENTOS METODOLÓGICOS

Neste capítulo, serão descritos todos os procedimentos realizados neste trabalho. Como a nova arquitetura da API *PreDetect* e seu fluxo de funcionamento. Também será apresentada uma aplicação de localização *indoor* que foi utilizada para validar o módulo BLE da API.

4.1 A arquitetura da API

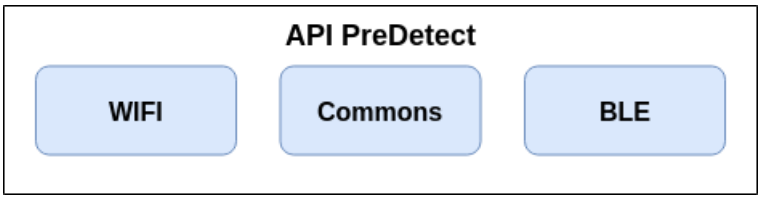
A API *PreDetect* possui uma arquitetura monolítica, sem modularidade interna. Para facilitar a adição de novas funcionalidades com contextos ou abordagens diferentes, mantendo a API desacoplada e com alta coesão, é necessário migrar para uma arquitetura modular.

A nova arquitetura permite a criação de módulos específicos para cada tipo de *data source*, ou seja, um módulo *WiFi* e outro *Bluetooth Low Energy*. Possibilitando ao desenvolvedor uma maior liberdade ao escolher somente ao o que atende a sua necessidade ou mesmo optar por uma abordagem híbrida.

Ao adotar uma arquitetura modular, surge a problemática de duplicação do código. Para amenizar esse problema foi criado um módulo com as configurações comuns entre os módulos.

Na Figura 6 é possível visualizar a arquitetura.

Figura 6 – Arquitetura Modular da API *PreDetect*



Fonte – Elaborada pelo autor.

4.2 Módulo *Bluetooth Low Energy*

Para a construção do módulo foi utilizado o suporte nativo da plataforma *Android* para o *Bluetooth Low Energy* (BLE). A versão utilizada foi a do *Android* 5.0 *Lollipop* (API *level* 21). Dessa forma, foi possível obter os dados das redes *Bluetooth Low Energy* próximas ao dispositivo.

Para poder utilizar o *Bluetooth Low Energy* é necessário algumas permissões do sistema. Também é necessário a permissão para acessar a localização por possuir *micro-location*. Segue as permissões:

```
<manifest>
    <uses−permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <uses−permission android:name="android.permission.BLUETOOTH" />
    <uses−permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses−feature android:name="android.hardware.bluetooth_le" android:required="true" />

    ...
</manifest>
```

Diferente da rede *WiFi* a rede *Bluetooth* no *Android* possui um gerenciador central que possibilita requisitar os dados de maneira simplificada. É necessário somente cadastrar um *callback* para realizar as ações desejadas quando encontrar algum sinal relacionado ao *Bluetooth Low Energy*.

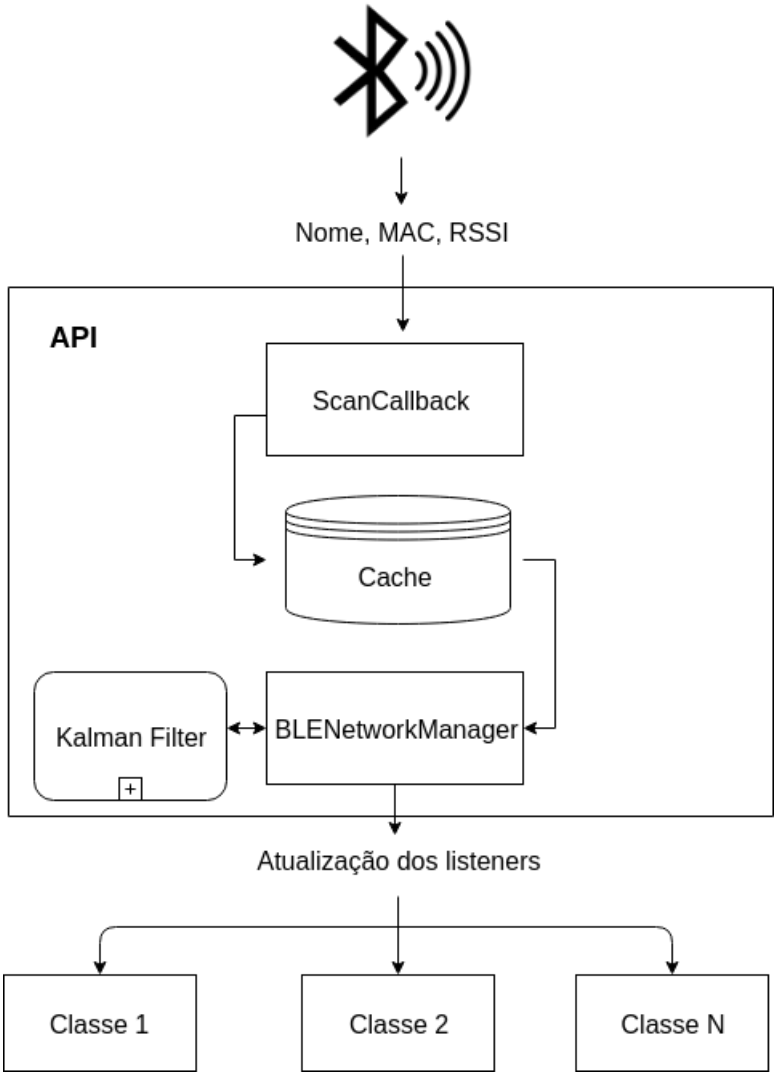
No *callback* o processo de obtenção dos dados das redes *BLE* se inicia. Para se obter os dados o *callback* tem o tipo *ScanCallback* e sobrescrever o método *onScanResult*, com ele tem acesso a classe *ScanResult* que contém o nome, MAC, RSSI, entre outros dados das redes BLE que não são relevantes para o módulo proposto.

Todos os dados são repassados para a classe *BLENetworkManager*, onde os mesmos são encapsulados em uma lista de objetos pertencentes a classe *Beacon*, essa última, possui os atributos nome, MAC, RSSI e *distance*. Quando um sinal é detectado ele fica armazenada em cache até que a sua quantidade alcance 100 amostras, quando esse valor é atingido, será utilizado a Fórmula 2.2 do *Kalman Filter* para melhorar a estimativa do RSSI, logo em seguida, a distância é calculada com esse RSSI estimado. Essa conversão é provida pela transformação do RSSI em distância utilizando a Fórmula 2.1.

Com o intuito de prover os dados para a aplicação de forma simplificada foi utilizado o padrão *Observer*, dessa forma a classe interessada deve implementar a interface *BeaconListener* e registrar seu interesse em observar o comportamento das redes BLE utilizando o método *registerListener* no *BLENetworkManager*.

Todo o processo exemplificado anteriormente pode ser visualizado na Figura 7

Figura 7 – Fluxo dos dados das redes *Bluetooth Low Energy*

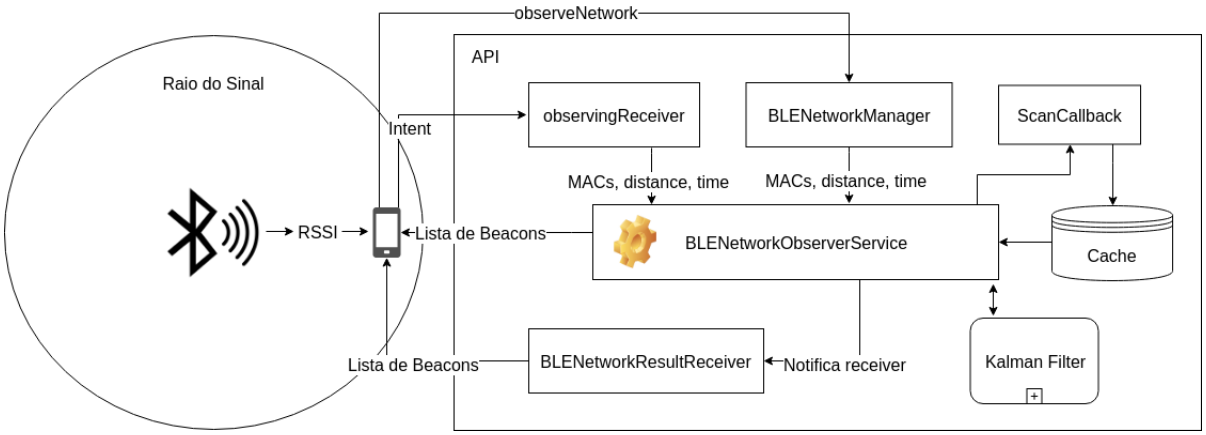


Fonte – Elaborada pelo autor.

Outra funcionalidade fornecida pelo módulo é de checar se um dispositivo está presente dentro de um raio de distância em um tempo especificado. Com essa funcionalidade se torna possível saber a porcentagem de tempo que um dispositivo ficou próximo de determinada rede *Bluetooth Low Energy*. Toda a sua implementação utiliza a classe *Service* fornecida pela a plataforma *Android*, portanto a sua execução será em segundo plano.

A Figura 8 apresenta o fluxo de funcionamento dessa funcionalidade.

Figura 8 – Fluxo da detecção de presença



Fonte – Elaborada pelo autor.

Existem duas maneiras de solicitar esse serviço. A primeira maneira é mandatório implementar a interface *BeaconObserver* e sobrescrever o método $onObservingEnds(networkResult : NetworkResult < Beacon >)$, que como argumento, tem como resultado os dados abstraídos em uma classe *wrapper* $NetworkResult$. No *wrapper* contêm as seguintes abstrações:

```
networkResult
    .onSuccess { list: List<Data>? ->
        // TODO
    }
    .onFail { list: List<Data>? ->
        // TODO
    }
    .onUndefinedNetwork {
        // TODO
    }
```

O parâmetro $list : List < Data >?$ representa a lista com os resultados da observação do dispositivo. Para cada objeto nessa lista é possível obter a porcentagem de tempo que o *smartphone* ficou próximo dos *Beacons BLE* usados como referência. Feito isso, uma instância da classe *BLENetworkManager* deve ser obtida sendo possível executar o método $observeNetwork(observer : BeaconObserver, btMACsToObserve : List < String > , timeInMinutes : Int, maxRangeInMeters : Double, intervalTimeInMinutes : Int)$, passando a

instância da classe que implementa *BeaconObserver*, uma lista de MACs que serão observados, em seguida, um inteiro informando o tempo em minutos que os MACs seriam observados, um raio de distância mínimo entre o *smartphone* e o *Beacons BLE* em metros e o intervalo entre as checagens em minutos.

A segunda maneira de solicitar esse serviço é lançando uma *Intent* com os dados do raio distância do *Beacon* para o *smartphone*, como também o tempo de checagem e por último uma lista de MACs de referência, o formato dos parâmetros obedecem a mesma assinatura do método utilizado pelo *BLENetworkManager*. Quando essa *Intent* é lançada o *BroadcastReceiver BLEObservingReceiver* é notificado. Este último inicia o serviço *BLENetworkObservingService* em segundo plano e repassa todos os dados vindos da *Intent*. Diferente da outra forma de requisição do serviço, nessa forma quem o inicia é a classe *BLEObservingReceiver* e não *BLENetworkManager*. Mas para que seja possível executar dessa maneira é necessário a configuração da classe *BLENetworkReceiver* no arquivo *AndroidManifest*. A configuração necessária é a seguinte:

```
1 <application>
2     ...
3     <service android:name="br.ufc.predetect.ble.services.BLENetworkObserverService" android:
          exported="true" android:enabled="true">
4         <intent-filter android:priority="100000">
5             <action android:name="br.ufc.predetect.ble.NETWORK_SERVICE" />
6             <category android:name="android.intent.category.DEFAULT"/>
7         </intent-filter>
8     </service>
9 </application>
```

A classe *BLENetworkObservingService* é simplesmente uma *thread* que, a partir de um determinado tempo configurado, captura os *Beacons BLE* próximos, verifica se o MACs informados estão entre os *Beacons* capturados, prontamente calcula a distância para cada um deles e verifica se o dispositivo está dentro da distância mínima informada. No final da verificação é retornado a percentagem de tempo que o dispositivo ficou presente dentro da distância especificada.

Realizar essas verificações frequentes dos dados consome muita energia, quando o *Android* identifica que o dispositivo não está em uso, é acionado um mecanismo de

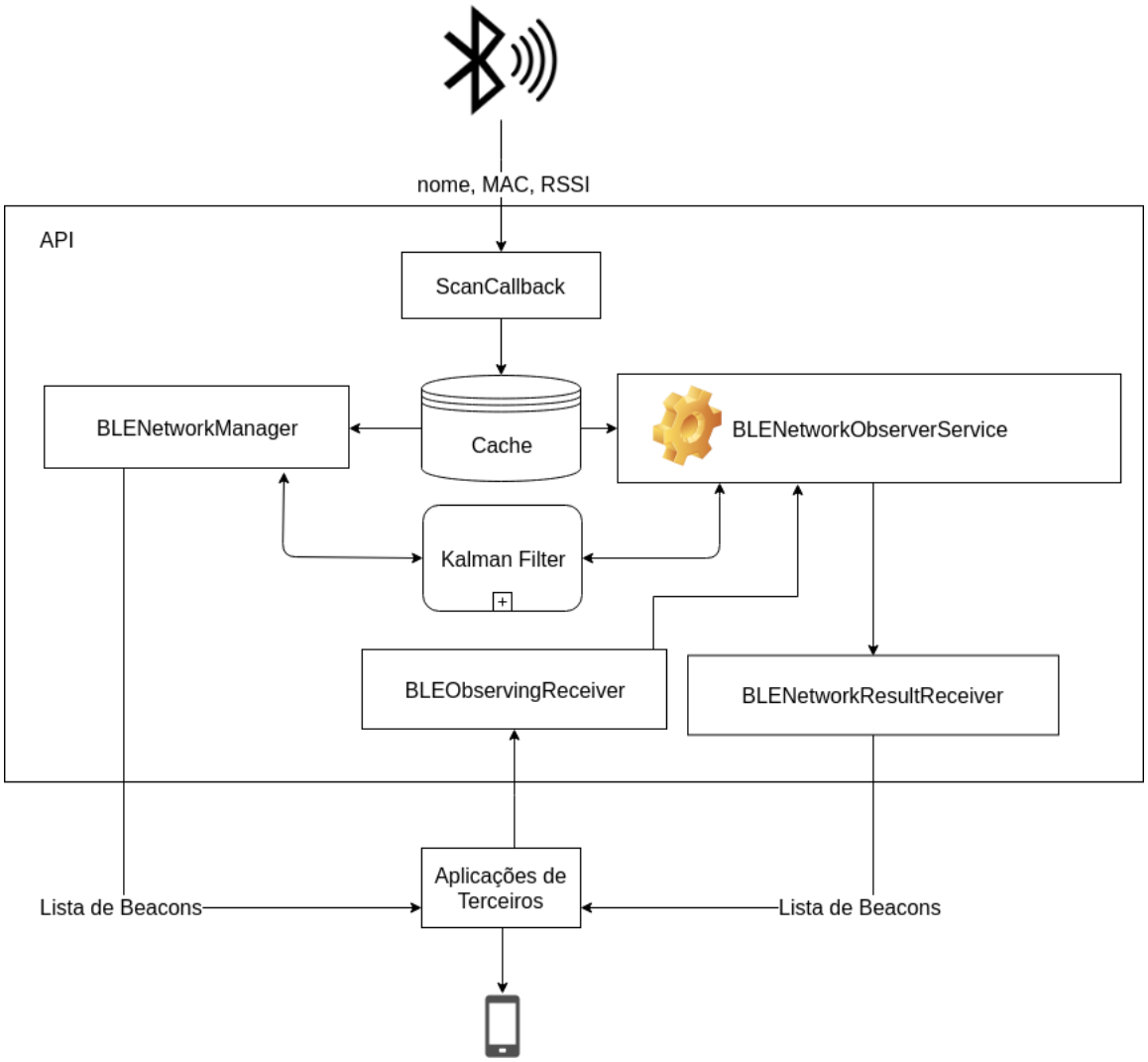
gerenciamento de bateria que desabilita todos os seus componentes, inclusive a CPU (KIM; CHA, 2013). Porém esse mecanismo deve ser utilizado com cuidado, em um cenário onde alguma aplicação esteja realizando *donwload* de algum conteúdo da internet, seu comportamento será afetado. Para evitar que durante o serviço de detecção de presença o dispositivo utilizado desabilite a CPU foi necessário o gerenciamento de um *WakeLock*. *WakeLock* é um mecanismo que garante que o dispositivo ficará ativo mesmo depois de muito tempo de inatividade quando a aplicação está executando.

Quando solicitado, o serviço de detecção de presença, implementado na API utilizará apenas o *WakeLock* que ativa a CPU. Portanto, mesmo se o dispositivo ficar sem a interação do usuário, a CPU e o BLE não serão desabilitados. Para evitar o consumo indevido de bateria, quando o serviço é finalizado, o *WakeLock* é liberado.

Deve se destacar que não é necessário o *smart phone* estar conectado a rede BLE, já que dados como nome, MAC, RSSI são disponibilizados sem haver a conexão entre os mesmos.

Uma visão geral da arquitetura da API com todas as funcionalidades descritas anteriormente, pode ser visualizada na Figura 9.

Figura 9 – Visão geral do módulo BLE



Fonte – Elaborada pelo autor.

4.2.1 Documentação do módulo BLE

A API está disponível no Github[1] com toda a documentação reescrita para atender as mudanças realizadas. Também foi adicionado um exemplo de como utilizar os módulos existentes. Todo código e sua documentação pode ser acessado em <https://github.com/gabrielczar/PreDetect>.

4.3 Desenvolvimento da aplicação de presença

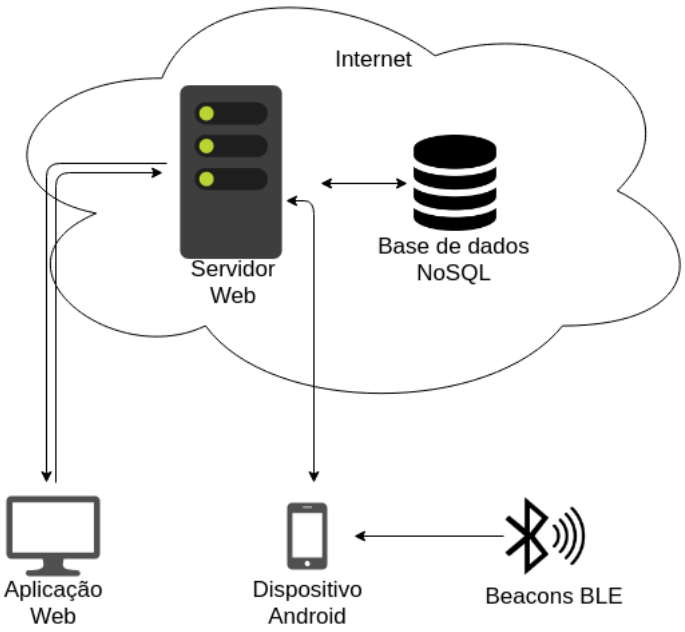
Como estudo de caso e para ser utilizado como validação do módulo foi criada uma aplicação responsável por identificar a presença de dispositivos em determinados horários. O

[1] https://github.com

desenvolvimento dessa aplicação foi dividido em duas etapas. A primeira etapa consiste no desenvolvimento de uma aplicação cliente para a plataforma *Android* e outra para a plataforma WEB, ambas aplicações serão abordadas nessa seção.

Na figura abaixo pode ser visualizado as partes existentes e como elas se comunicam:

Figura 10 – Comunicação entre as partes da aplicação.



Fonte – Elaborada pelo autor.

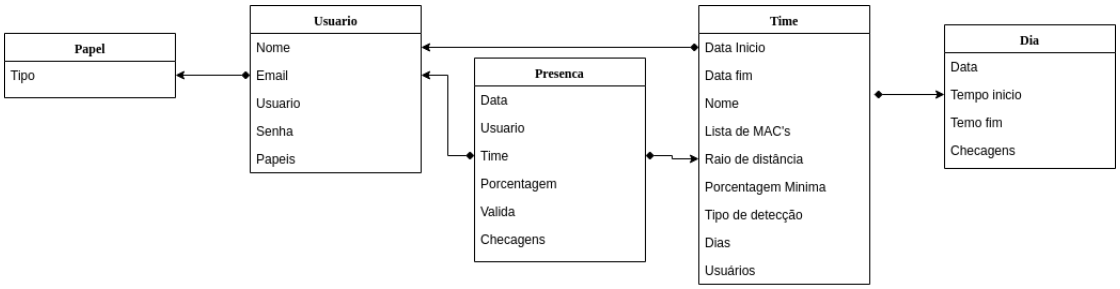
Com base na Figura 10 é possível visualizar como a aplicação está dividida: uma aplicação móvel, onde o usuário terá acesso a todas as suas turmas e também onde a sua presença será computada; a aplicação Web onde é possível cadastrar turmas e horários de checagem da presença; servidor web onde todas as requisições serão tratadas e salvas em uma base de dados *NoSQL*; *Beacons* BLE que serão utilizados como pontos de referência na checagem de presença da aplicação cliente.

4.3.1 Aplicação Servidor

O servidor Web é o responsável por responder as requisições de outras aplicações permitindo o acesso aos dados salvos na base de dados. A aplicação servidor foi criado por Filho (2016), utilizando *NodeJS* e será utilizado nesse trabalho com algumas modificações para atender aos requisitos.

Na Figura 11, o mapeamento das entidades pode ser visualizado.

Figura 11 – Entidades do banco de dados.



Fonte – Elaborada pelo autor.

Nessa figura é possível observar todas as entidades modeladas para a aplicação de presença. Cada uma dessas entidades são representadas em forma de documentos no banco de dados. A entidade Usuário possui apenas dados básicos e contém os Papéis com as suas responsabilidades. Os papéis padrão são Administrador e Estagiário. Um administrador é responsável pelo gerenciamento das turmas e dos horários de checagem de presença. Já um estagiário pode somente, olhar os dados das turmas que está presente ou visualizar sua presença (FILHO, 2016).

Um time tem um período de atividades especificado, como também uma lista de estagiários que o compõem. Além do período também é necessário informar os dias de trabalho desse time. A entidade Dia possui os horários de início e fim de um dia específico de trabalho, como também os horários e durações das checagens de presença, que são utilizados pela a aplicação móvel (FILHO, 2016). Além dos dias, um time possui uma lista de MACs que representam os pontos de referências que serão utilizados para realizar a checagem de presença na aplicação móvel. Cada time possui um raio de distância que os dispositivo móvel poderá ficar para que suas presenças sejam confirmadas, além de porcentagem que informará se o estagiário está presente ou não. Também deve ser informado o tipo de detecção que será utilizado para realizar a checagem da presença, no momento, existem as seguintes opções: *WIFI*, *BLE* e *HYBRID* (híbrida).

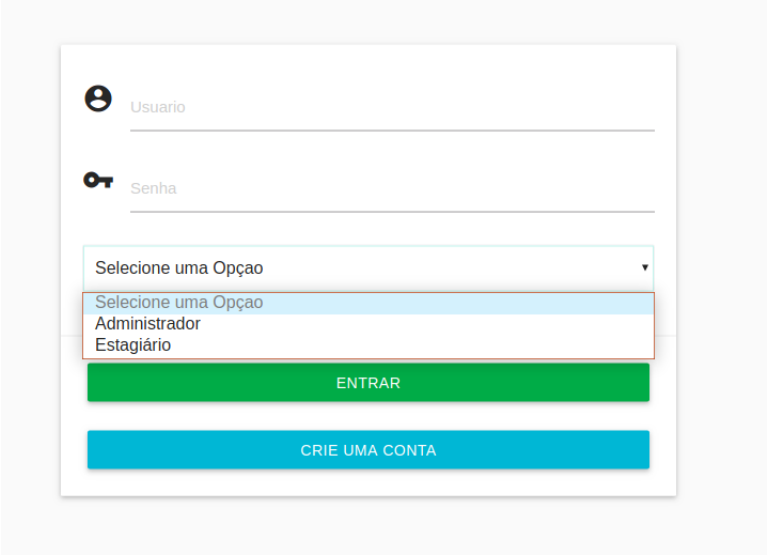
A entidade Presença é utilizada para registrar as presenças de cada estagiário, essa entidade possui atributos como data da presença, o usuário (estagiário) e o time vinculado a ela, uma quantidade e percentagem de checagens que são feitas no dia e por fim a validade dessa presença, que informa se a média da presença é maior ou igual a porcentagem cadastrada no time (FILHO, 2016).

4.3.2 Aplicação Web

Uma aplicação Web foi desenvolvida para para oferecer uma interface amigável para o acesso dos dados no servidor. Assim como o servidor, essa aplicação também foi reaproveitada e foram realizadas algumas modificações para se adequar a nova modelagem. Para o desenvolvimento dessa aplicação, Filho (2016) utilizou o *framework AngularJs* junto com CSS e HTML para a estruturação das páginas. *AngularJs* é um *framework* de desenvolvimento de aplicações Web sob o padrão MVC (*Model – View – Controller*) utilizando a linguagem *Javascript*, sendo uma plataforma de código aberto mantida pelo *Google*.

As interfaces principais são as seguintes: uma página de autenticação que redireciona o usuário baseados nos seus papéis, na Figura 12 é visualizada essa interface. Diferente do administrador o estagiário possui apenas funcionalidades básicas como se cadastrar e visualizar as turmas em que está presente.

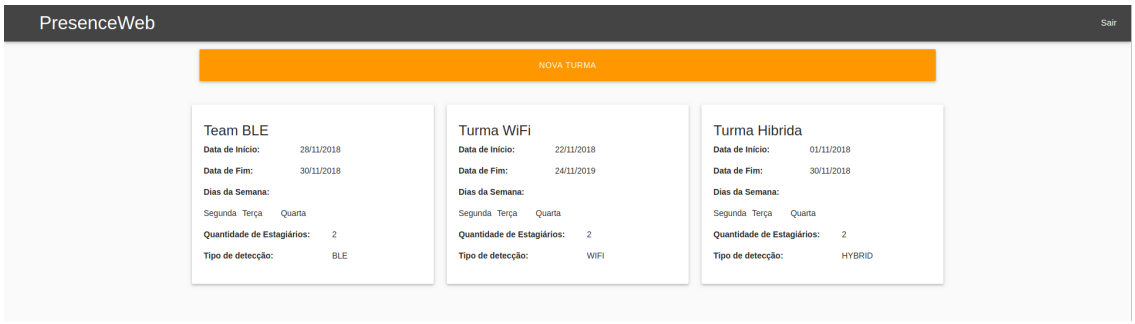
Figura 12 – Tela de autenticação.



Fonte – Elaborada pelo autor.

Quanto ao administrador, a aplicação permite visualizar os times existentes e as informações básicas de cada um, como pode ser visto na Figura 13.

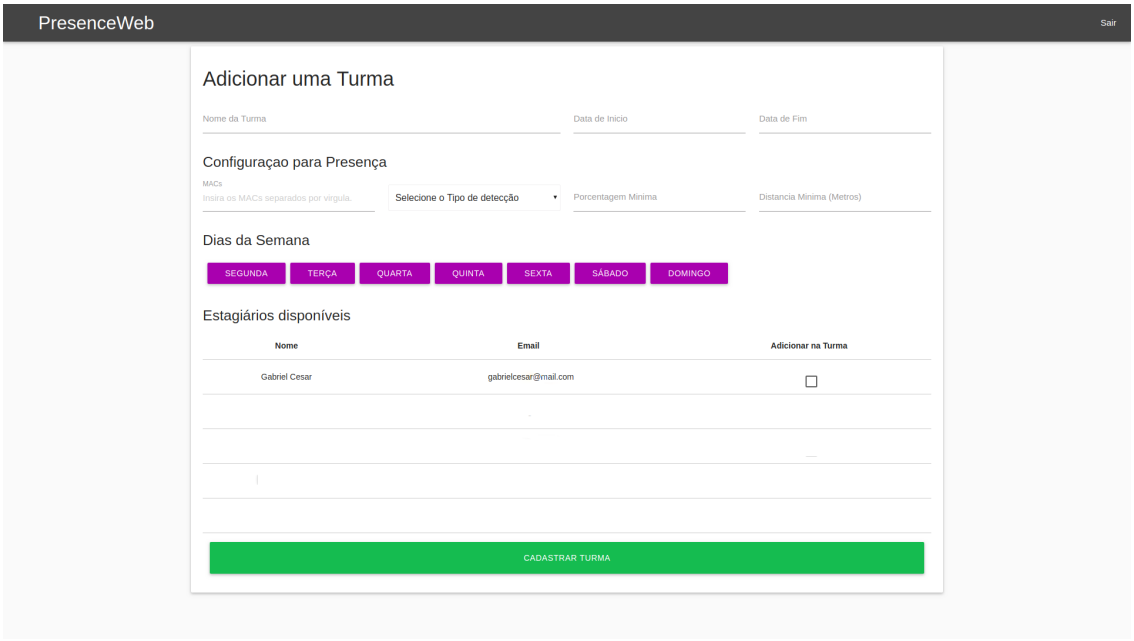
Figura 13 – Tela de Administração.



Fonte – Elaborada pelo autor.

O administrador ao clicar em nova turma poderá cadastrar uma nova turma e suas configurações, como MACs para checagem, tipo de detecção, percentagem de presença miníma e distância miníma permitida.

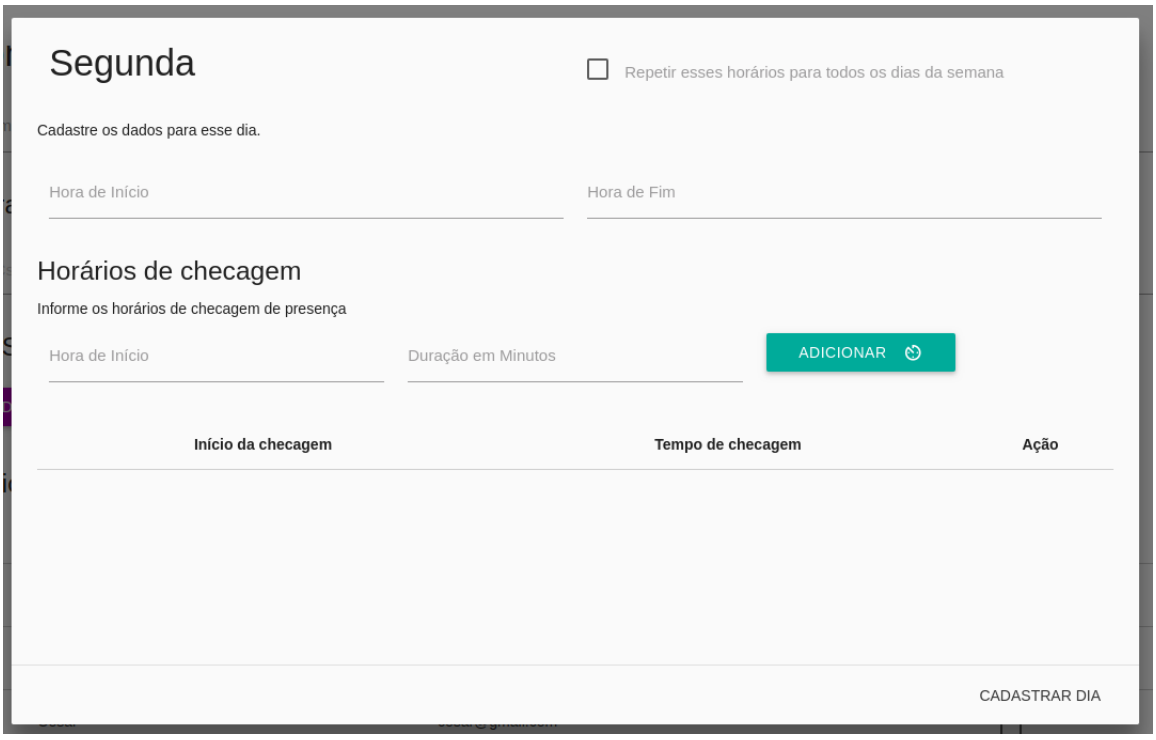
Figura 14 – Tela de Cadastro de Turmas.



Fonte – Elaborada pelo autor.

Além desses dados o administrador deve cadastrar os horários de atividade do time para cada dia da semana. Quando o administrador clica em um dia, um modal é aberto. Esse modal pode ser visualizado na Figura 15.

Figura 15 – Modal de Cadastro de Dias.



Fonte – Elaborada pelo autor.

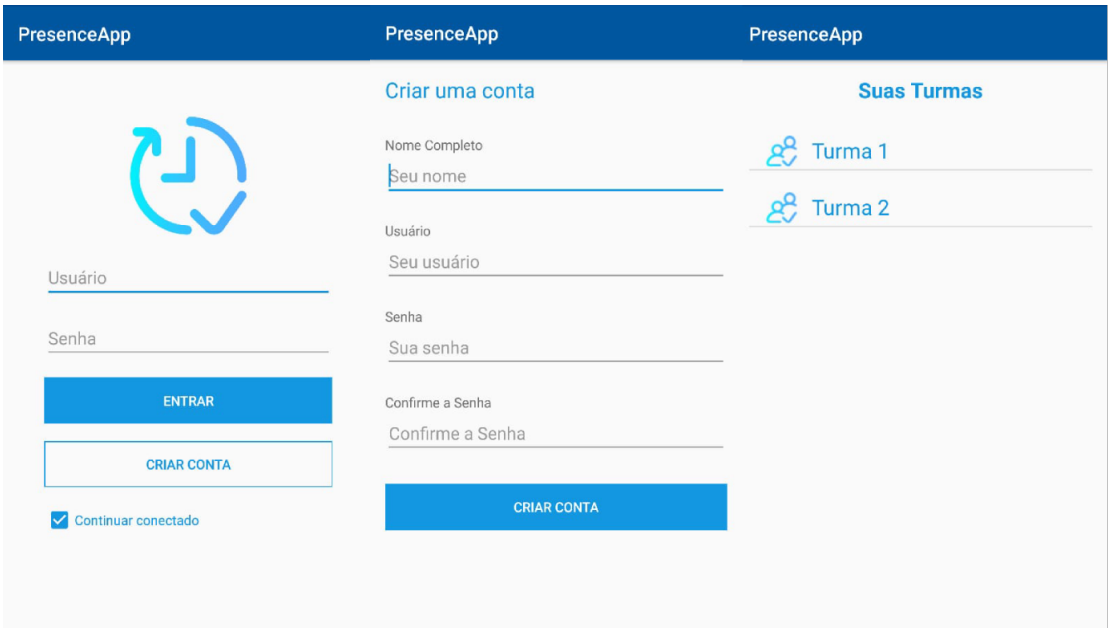
Nesse modal é possível cadastrar os horários de início e de fim de atividades para o dia escolhido. Logo depois é necessário informar os horários de checagem de presença, indicando a hora de inicio da checagem e a duração dessa checagem. Essas checagens serão utilizadas na aplicação móvel para determinar se o estagiário está presente.

4.3.3 Aplicação Móvel

Essa seção irá apresentar a aplicação desenvolvida para a plataforma *Android* utilizando a API desenvolvida. Serão apresentadas as telas principais e seu modo funcionamento.

Antes de abordar o funcionamento interno da aplicação, as telas principais serão apresentadas.

Figura 16 – Telas da aplicação móvel.



Fonte – Elaborada pelo autor.

A primeira tela representa a tela de autenticação, é nessa tela onde o estagiário informa suas credenciais, se ainda não possuir um conta, é possível realizar o cadastro clicando no botão "Criar Conta", que irá levá-lo para a segunda tela. Na segunda tela o estagiário poderá informar todos os seus dados. Quando sua conta é criada, o mesmo é direcionado para tela inicial onde todas as suas turmas são apresentadas.

4.3.4 Arquitetura da Aplicação

Toda a comunicação entre as aplicações e feita utilizando objetos JSON. Também foi elaborado um padrão para as repostas do servidor para facilitar a comunicação entre as partes. A propriedade *result* pode assumir dois valores: *true* ou *false*, esses valores informam se o processamento realizado pelo servidor foi executado com sucesso. A propriedade data representa um objeto ou uma lista de objetos retornados pelo servidor (FILHO, 2016).

```
1 { result: true, data: {} }
```

Quando o servidor retorna as turmas, é agendado os horários de checagem de presença. Esse agendamento é feito utilizando uma classe *AlarmManager*, possibilitando agendar a execução de um processamento no futuro. Para isso uma *Intent* é utilizada para enviar uma mensagem ao sistema informando a data e hora desejada (LECHETA, 2013).

Quando o alarme é disparado, o serviço de observação de presença da API é executado. No fim da execução do serviço, outra *Intent* é lançada no sistema, informando a porcentagem de presença do dispositivo móvel em relação ao ponto de referência. A aplicação de presença está configurada para receber essa *Intent*. Logo essa porcentagem é adquirida pela aplicação de presença e enviada para o servidor. Quando essa porcentagem é enviada para o servidor é feita uma verificação, se a checagem de presença recebida é a última do dia uma média de presença é calculada e a partir dessa média é possível saber se a presença do estagiário é válida. Caso não seja a última checagem do dia, o servidor responde informando que a checagem foi salva e está pronto para receber a próxima. Ao final é enviada um notificação ao usuário informando se a presença foi confirmada ou não (FILHO, 2016).

5 RESULTADOS E DISCUSSÕES

Este Capítulo irá abordar todos os resultados obtidos com o desenvolvimento e teste da API.

Utilizando a API foi possível verificar o comportamento das ondas do *Beacon BLE*, assim como a precisão do cálculo da distância para o mesmo. Foi elaborado alguns testes para checar a variação do indicador de força do sinal recebido (RSSI) e consequentemente a variação do cálculo da distância.

O *smartphone*, com uma aplicação de teste que utiliza o módulo desenvolvido, foi colocado em uma posição fixa em uma sala, à 1 metro e a 3 metros de distância do *Beacon BLE* utilizado como ponto de referência. Durante a coleta nenhum obstáculo impediu a passagem do sinal.

Todos os dados coletados foram armazenados em um arquivo no formato CSV (*Comma Separated Values*). No final da coleta, os dados foram postos em uma planilha para esboçar os gráficos para facilitar a visualização. Uma amostra dos dados coletados pode ser visualizada na Figura 17.

Figura 17 – Amostra dos dados coletados

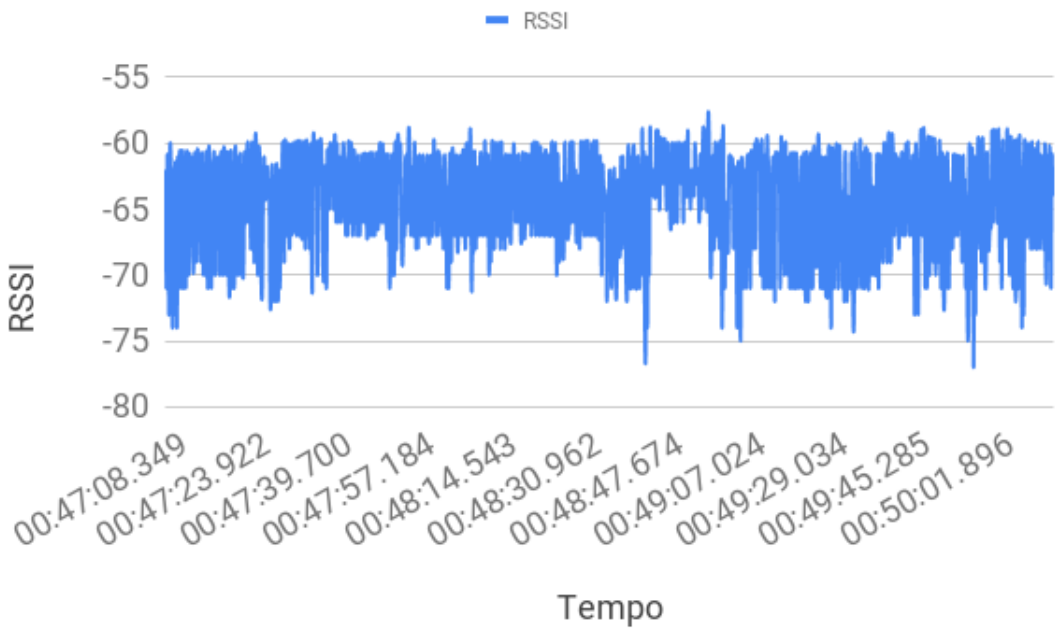
MAC	RSSI	TEMPO	NOME	DISTANCIA
F0:C7:7F:EB:89:5E	-82	05:43:49.265000	HMSoft	3.3
F0:C7:7F:EB:89:5E	-82	05:43:50.218000	HMSoft	3.3
F0:C7:7F:EB:89:5E	-86	05:43:51.298000	HMSoft	4.1
F0:C7:7F:EB:89:5E	-88	05:43:52.207000	HMSoft	4.7
F0:C7:7F:EB:89:5E	-85	05:43:53.244000	HMSoft	3.9

Fonte – Elaborada pelo autor.

Nessa planilha são armazenados o MAC do *Beacon BLE*, o indicador de força do sinal recebido (RSSI), o horário da coleta, o nome do *Beacon BLE* e a distância calculada pela API no momento da coleta. A cada segundo a API coletou dados de todos os *Beacons BLE* próximos. No entanto, por limitações do *Bluetooth Low Energy* da plataforma *Android* é necessário ter alguns intervalos entre consultas longas para não concorrer com outros serviços executados no sistema. O valor encontrado foi de 12 segundos que possibilitou realizar testes por vários minutos sem interferências.

No gráfico a seguir, é visualizado 1000 amostras coletadas durante 3 minutos. São coletadas 100 amostras e espera-se 12 segundos para coletar a próxima amostra.

Figura 18 – Amostra dos dados coletados a 1 metro de distância.

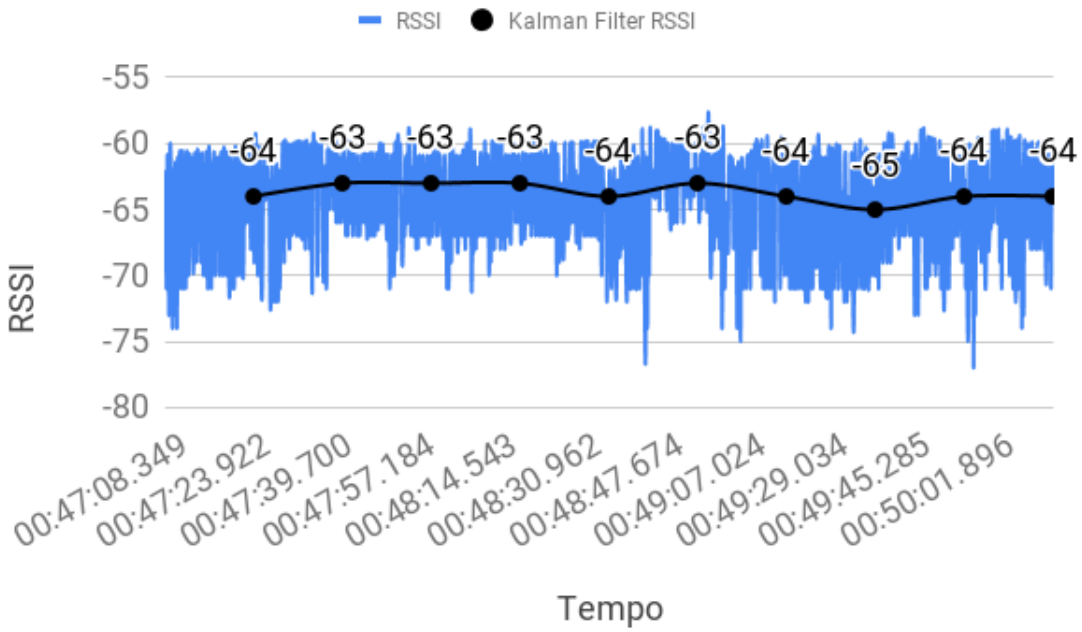


Fonte – Elaborada pelo autor.

É possível notar no gráfico apresentado que o RSSI varia bastante mesmo o *smartphone* estando em uma posição fixa, além da ausência de barreiras entre ele o *Beacon BLE*. Essa variação afeta diretamente na precisão da distância fornecida pela API desenvolvida. O valor esperado para o RSSI a 1 metro de distância é aproximadamente -62 utilizando a Formula 2.1.

Para melhorar essa estimativa do RSSI foi utilizado a Formula 2.2 do *Kalman Filter*, e o resultado pode ser visualizado no gráfico a seguir.

Figura 19 – Amostra dos RSSIs coletados a 1 metro de distância com a aplicação do *Kalman Filter*.

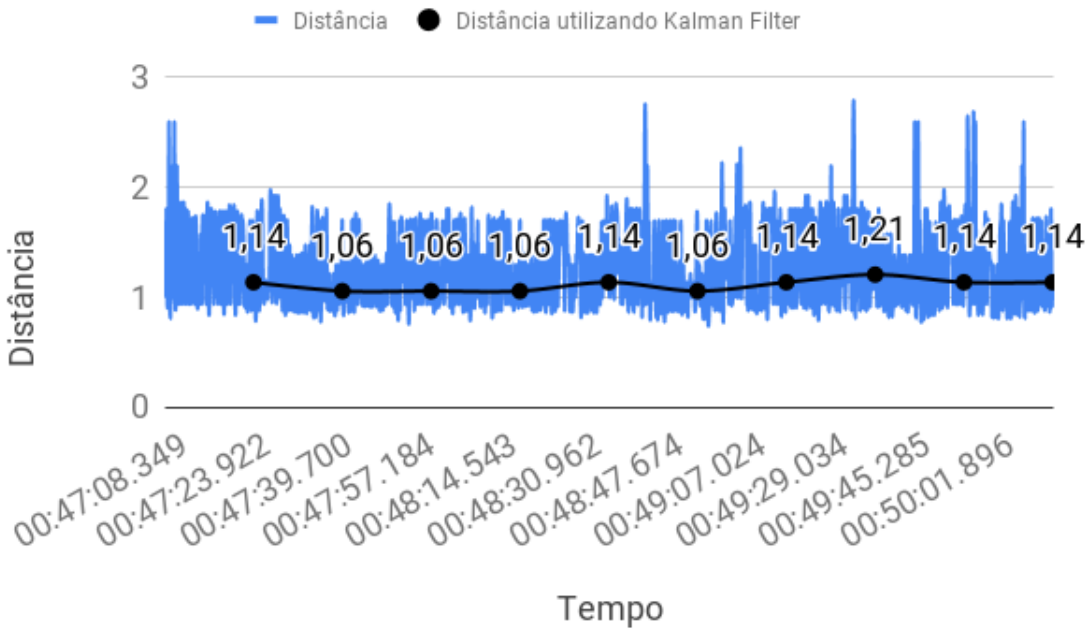


Fonte – Elaborada pelo autor.

São coletadas 100 amostras e logo em seguida é aplicado o *Kalman Filter* nessas amostras. Como resultado foi obtido um RSSI que varia de -63 até -64 nesses 3 minutos, bem mais preciso e próximo do valor esperado.

Na gráfico a seguir pode ser visualizado os resultados utilizando as distâncias.

Figura 20 – Amostra das distâncias coletados a 1 metro de distância com a aplicação do *Kalman Filter*.

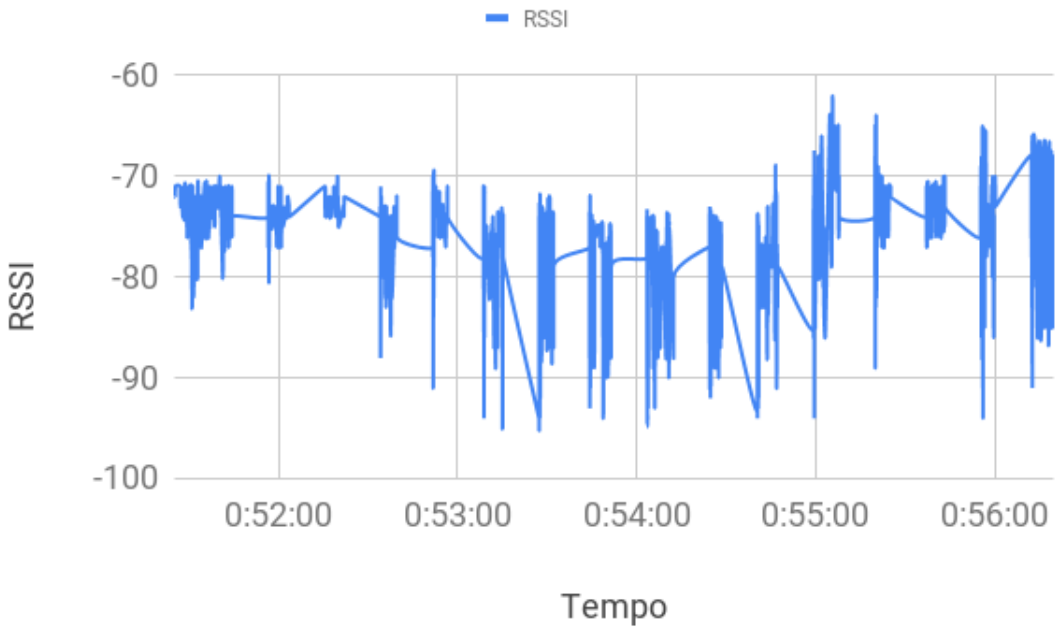


Fonte – Elaborada pelo autor.

Com a Figura 20 é possível visualizar mais precisamente a diferença, foram obtidas variações muita próximas da esperada a margem de erro nesta amostra foi de aproximadamente 22 centímetros. Resultado esse que comparado com os valores obtidos pela mesma distância entre um dispositivo e um WiFi utilizado como ponto de referência, foi bem mais preciso, pois o módulo WiFi obteve uma margem de erro de aproximadamente 73 centímetros.

Durante os testes, agora com 3 metros de distância entre *smartphone* e o *Beacon BLE*. Foram coletadas 1600 amostras durante 5 minutos. São coletadas 100 amostras e espera-se 12 segundos para coletar a próxima amostra.

Figura 21 – Amostra dos dados coletados a 3 metros de distância.

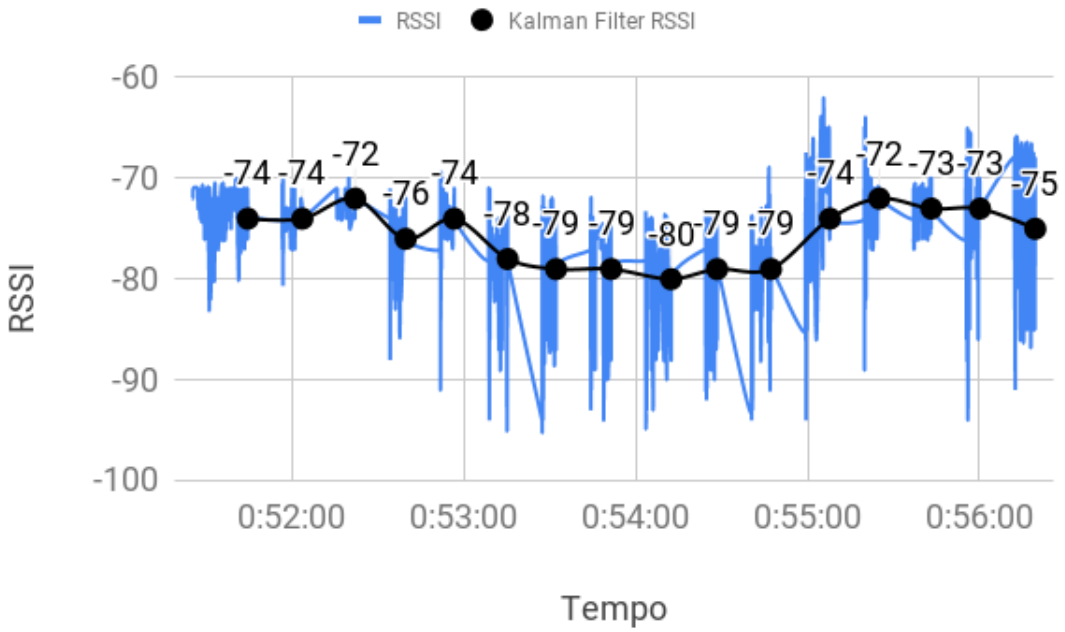


Fonte – Elaborada pelo autor.

Com o esboço do gráfico fica visível a grande variação do RSSI quando aumenta a distância até o *Beacon BLE*. Assim como o teste a 1 metro, o dispositivo permaneceu em uma posição fixa, além da ausência de barreiras. O valor esperado para o RSSI a 3 metros de distância é aproximadamente -83 utilizando a Formula 2.1.

Utilizando a Formula 2.2 do *Kalman Filter* para melhorar a estimativa obteve o seguinte resultado:

Figura 22 – Amostra dos RSSIs coletados a 3 metros de distância com a aplicação do *Kalman Filter*.

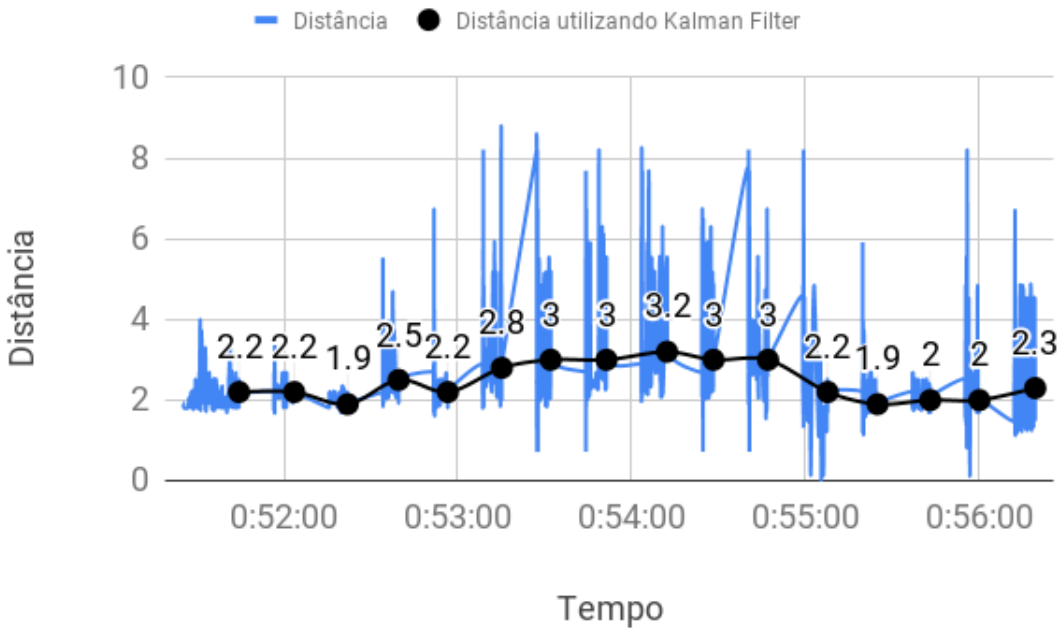


Fonte – Elaborada pelo autor.

Assim, como a aplicação do *Kalman Filter* nas amostras de 1 metro, houve uma maior estabilização do RSSI, no entanto a variação que foi estimado está entre -72 e 80, enquanto a esperada é -83.

No gráfico a seguir é possível compreender melhor pois utiliza unidades de medida.

Figura 23 – Amostra das distâncias coletados a 3 metros de distância com a aplicação do *Kalman Filter*.



Fonte – Elaborada pelo autor.

Com a Figura 23 é possível visualizar mais precisamente a diferença, os valores variam de 1,93 metros a 3,27 metros, portanto a margem de erro nesta amostra foi de aproximadamente 107 centímetros.

Considerando as precisões encontradas é admissível afirmar que a utilização do BLE para distâncias próximas é mais adequada que utiliza-la para longas distâncias, pois se mostrou mais precisa. Para longas distâncias é mais adequado utilizar o módulo WiFi pois em testes realizados no trabalho de Filho (2016), foi encontrado uma boa aproximação da distância real.

Contudo, quando se deseja buscar por dispositivos utilizando o *Bluetooth* espera-se que ocorra rapidamente por ser um processo que demanda a utilização de muitos recursos do adaptador *Bluetooth*. Em testes utilizando o *Bluetooth* de forma exaustiva, encontrou-se um *gap* (espaço vazio) entre as buscas por dispositivos durante 3 horas consecutivas, ou seja, o adaptador encontrou dispositivos normalmente durante 1 hora e após esse período, passava 1 hora sem receber qualquer informação e posteriormente começava a receber as informações novamente.

Deste modo, foi considerado como uma forma de avaliar a eficiência da biblioteca, realizar essa busca durante 3 horas consecutivas com o menor *delay* (atraso) possível entre as buscas e que ocorresse sem grandes oscilações na quantidade de memória RAM utilizada pela

biblioteca.

O *delay* encontrado foi de aproximadamente 12 segundos entre cada busca por novos dispositivos, com isso o *Bluetooth* conseguiu manter a estabilidade durante as 3 horas. Na Figura 24, é possível visualizar que a CPU é utilizada rapidamente e depois espera a próxima chamada. Durante todo esse processo, a quantidade de memória utilizada se mantém quase constante em 9,7 MB. Assim sendo, é plausível afirmar que a biblioteca consegue executar por ao menos 3 horas sem maiores interrupções. Esses dados foram visualizados através do *Android Profiler*, ferramenta disponibilizada pelo *Android Studio*, onde é possível visualizar a quantidade de memória e também da CPU utilizada pela biblioteca.

Figura 24 – Quantidade de memória e CPU utilizada pela biblioteca.

Fonte – Elaborada pelo autor.

Para validar a eficácia do módulo foi utilizada a aplicação de presença. Foi posicionado um *Beacon* BLE em uma sala e a aplicação foi instalada em 3 *smartphones*. O MAC desse ponto foi registrado, junto com os dias de atividade na semana, como também os horários de checagem de presença para cada dia, um raio de distância de dez metros, e uma porcentagem mínima para validar a presença.

A Tabela 1 apresenta checagens de presença de um dia com os 3 participantes selecionados. Dois horários de checagem foram cadastrados, cada um com um tempo de duração de trinta minutos. No final da checagem, é calculado a média das checagens para cada participante. Todos os participantes estiveram presentes durante todo o tempo de checagem. Os *smartphones* utilizados possuíam respectivamente o *Android* 5.0, 7.0 e 8.0.

Tabela 1 – Checagens de presenças.

Participantes / Horários de checagem	09:00	10:00	Total
Participante 1	100%	100%	100%
Participante 2	0%	0%	0%
Participante 3	80%	100%	90%

Fonte – Elaborado pelo autor.

Nesse teste, todos os participantes estavam presentes em todas as checagens, porém, como pode ser visto na Tabela 1, mesmo presente, o Participante 2 não obteve sua presença. Logo, após esse acontecimento foi realizados testes diretamente com a API e aconteceu esse mesmo caso, no entanto, após reiniciar o *Bluetooth Low Energy* manualmente a aplicação voltou a funcionar corretamente. De certo modo, o módulo BLE funcionou como esperado mas ainda não possui contramedidas para problemas nativos da API *Bluetooth* do *Android*. O participante 3 não conseguiu realizar completamente todas as presenças mesmo presente no local, essa situação não conseguiu ser reproduzida novamente, mas existem possíveis causas como o *smartphone* ter entrado em modo de espera, onde a CPU foi desabilitada, e mesmo a API utilizando *WakeLocks*, o dispositivo não ter reagido ou mesmo por questões de redução de consumo de bateria, pois o *Android* 8.0 introduziu novas regras para aumentar a eficiência energética.

6 CONSIDERAÇÕES FINAIS

Os ambientes *indoor* promove uma gama de oportunidades para o desenvolvimento de novas aplicações, especialmente pelo fato das pessoas transitarem na maior parte do seu tempo nesse tipo de ambiente.

Esse trabalho apresentou um módulo BLE para a API *PreDetect* para auxiliar outros desenvolvedores na criação de novas aplicações para ambientes *indoor*. Com o desenvolvimento desse trabalho, ficou perceptível a aplicabilidade dos pontos de referência utilizando *Bluetooth Low Energy* para localização em ambientes *indoor* desfrutando da vantagem da grande variedade de dispositivos compatíveis.

Os dados coletados sobre a variação da força do sinal recebido, pode-se concluir que a precisão do cálculo da distância fornecida pela API não possui uma acurácia grande para longas distâncias. Sendo necessário, para aplicações mais críticas, o uso de outra técnica de posicionamento como triangulação ou análise de cena.

Ainda há bastantes pontos a serem melhorados na API. Um deles é a utilização de outros filtros para observar se há uma melhora na acurácia do cálculo das distâncias. Outro fator é realizar mais de uma tentativa caso não encontre o dispositivo. Utilização de outras técnicas de localização. Adicionar mecanismos para contornar possíveis limitações da API *Bluetooth Low Energy* nativa da plataforma *Android*.

Também é recomendado, a adição de novas funcionalidades, pois no momento não existe uma maneira prática para realização de eventos ao encontrar determinado ponto de referência, por exemplo, realizar alguma ação quando um ponto de referência é encontrado ou mesmo quando estiver a um certo alcance ou até mesmo em um intervalo de tempo estabelecido.

REFERÊNCIAS

ARGENOX TECHNOLOGIES. **Android 5.0 Lollipop brings BLE Improvements**. 2018. Disponível em: http://www.argenox.com/blog/android-5-0-lollipop-brings-ble-improvements/. Acesso em: 20 nov. 2018.

BEKKELIEN, A.; DERIAZ, M.; MARCHAND-MAILLET, S. **Bluetooth indoor positioning**. Dissertação (Mestrado) — University of Geneva, 2012.

BLUETOOTH SPECIAL INTEREST GROUP. **Specification of the Bluetooth System-Covered Core Package version: 4.0**. [S.l.], 2010. Acesso em: 29 mar. 2018.

BLUETOOTH SPECIAL INTEREST GROUP. **Radio Versions**. 2018. Disponível em: https://www.bluetooth.com/bluetooth-technology/radio-versions. Acesso em: 1 abr. 2018.

BRAHLER, S. Analysis of the android architecture. **Karlsruhe institute for technology**, v. 7, n. 8, 2010.

BULTEN, W.; ROSSUM, A. C. V.; HASELAGER, W. F. Human slam, indoor localisation of devices and users. In: IEEE. **Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on**. [S.l.], 2016. p. 211–222.

ERIK HELLMAN. **Bluetooth Low Energy on Android**. 2018. Disponível em: https://hellsoft.se/bluetooth-low-energy-on-android-part-1-1aa8bf60717d. Acesso em: 20 nov. 2018.

ESTIMOTE. **Estimote Indoor SDK for iOS**. 2018. Disponível em: https://github.com/Estimote/iOS-Indoor-SDK. Acesso em: 25 nov. 2018.

FILHO, F. L. D. **Uma API para Detecção de Presença de Dispositivos Móveis em Ambientes Indoor**. 2016. Monografia (Bacharel em Sistemas de Informação), UFC (Universidade Federal do Ceará), Quixadá, Brasil.

GOOGLE. **Arquitetura da plataforma**. 2018. Disponível em: https://developer.android.com/guide/platform/index.html. Acesso em: 22 abr. 2018.

KIM, K.; CHA, H. Wakescope: runtime wakelock anomaly management scheme for android platform. In: IEEE PRESS. **Proceedings of the Eleventh ACM International Conference on Embedded Software**. [S.l.], 2013. p. 27.

KUROSE, J. F.; ROSS, K. W. **Rede de computadores e a Internet**: uma abordagem topdown. 3.ed. tradução arlete suimille marquies. revisão técnica wagner luiz zucchi. [S.l.]: São Paulo: Pearson Addison Wesley, 2006. 625 p.

LAIRD. **Specification of the Bluetooth System-Covered Core Package version: 4.0**. 2013. Disponível em: http://www.summitdata.com/blog/ble-overview/. Acesso em: 1 abr. 2018.

LECHETA, R. R. **Google Android**: Aprenda a criar aplicações para dispositivos móveis com o android sdk. 3ª edição. [S.l.]: Novatec Editora, 2013.

MINDELL, D.; MOSS, F. **How an Inventor You've Probably Never Heard of Shaped the Modern World**. 2018. Disponível em: https://www.technologyreview.com/s/602287/how-an-inventor-youve-probably-never-heard-of-shaped-the-modern-world/. Acesso em: 11 dez. 2018.

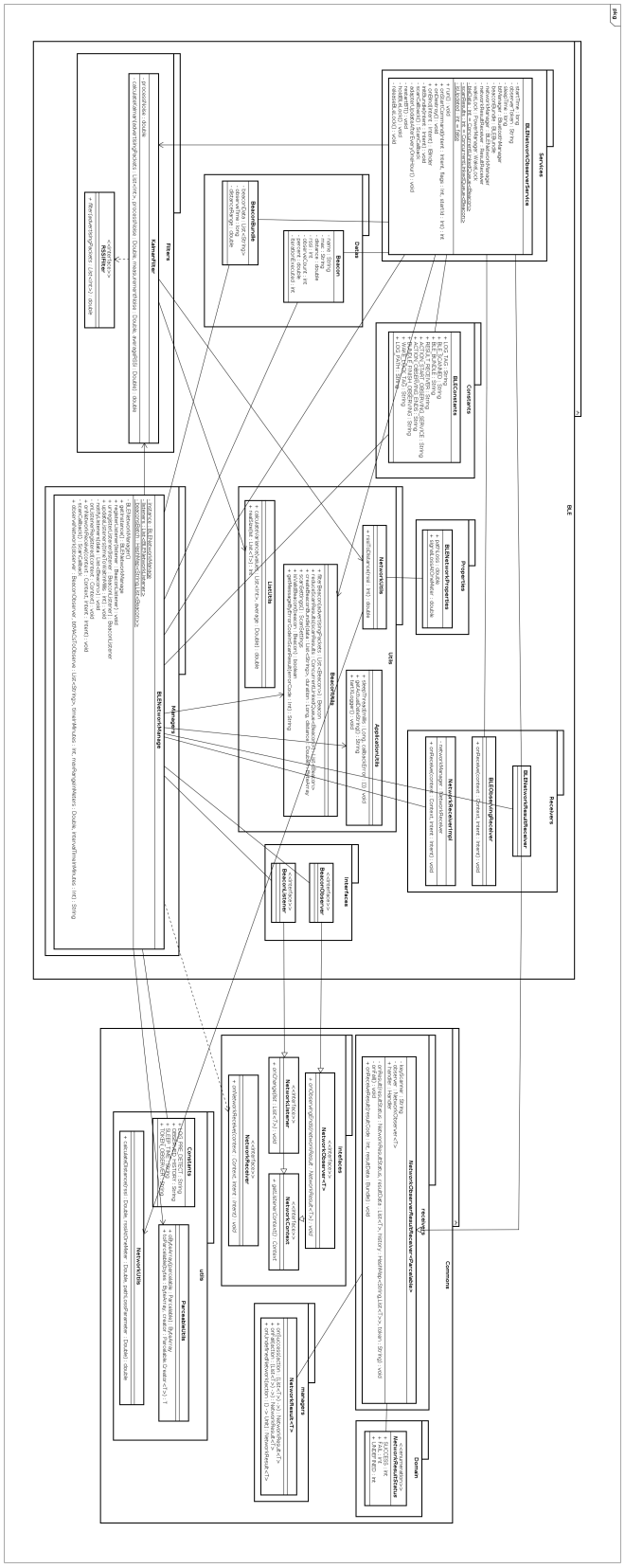
NEXENIO. **BLE-Indoor-Positioning**. 2018. Disponível em: https://github.com/neXenio/BLE-Indoor-Positioning. Acesso em: 25 nov. 2018.

SIMÕES, D. M. **Navegação indoor baseada na rede WIFI como suporte a serviços baseados na localização: estudo de caso no campus da UL**. Tese (Doutorado) — Universidade de Lisboa (UL), 2015.

THACKER, N.; LACEY, A. **Tutorial**: The kalman filter. [S.l.]: Citeseer, 1998.

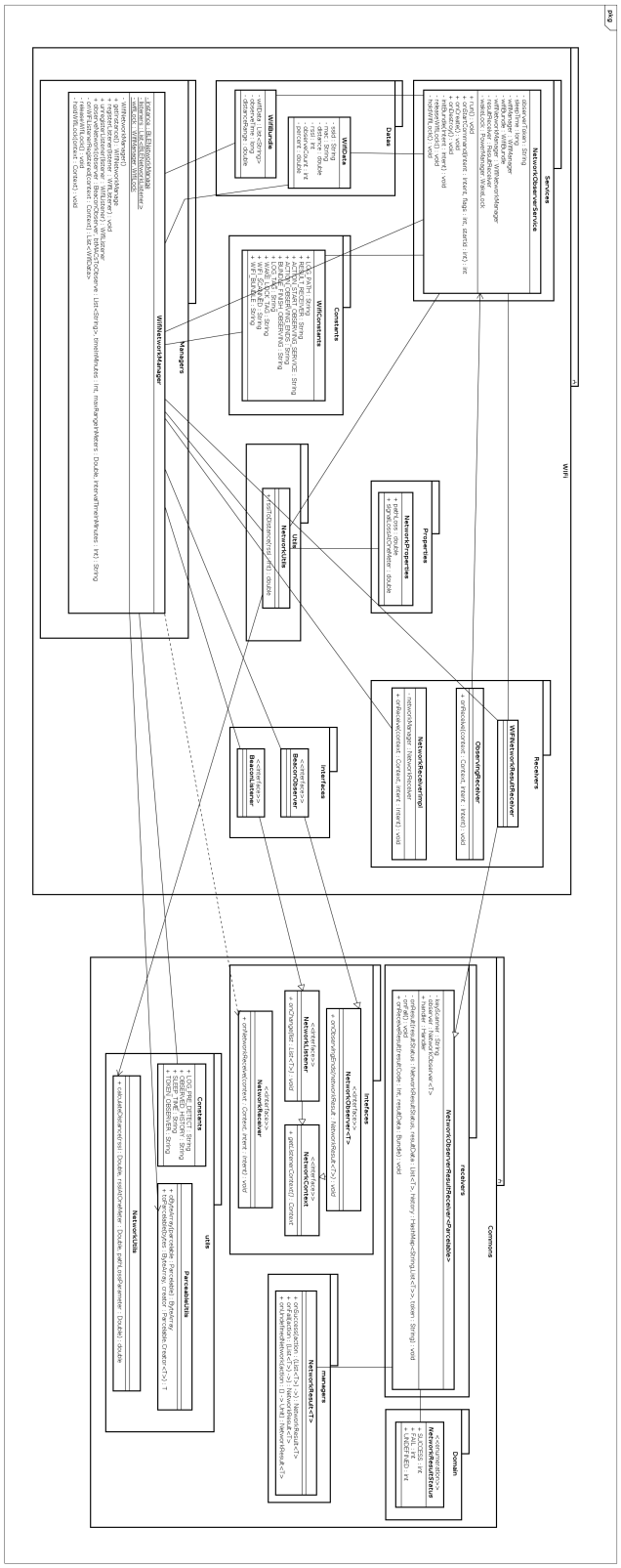
ZAFARI, F.; PAPAPANAGIOTOU, I. Enhancing ibeacon based micro-location with particle filtering. **IEEE**, San Diego, CA, USA: Global Communications Conference (GLOBECOM), 2015 IEEE, 2015. p. 1–7.

APÊNDICE A – DIAGRAMA DE CLASSE DA API PREDETECT MÓDULO BLE



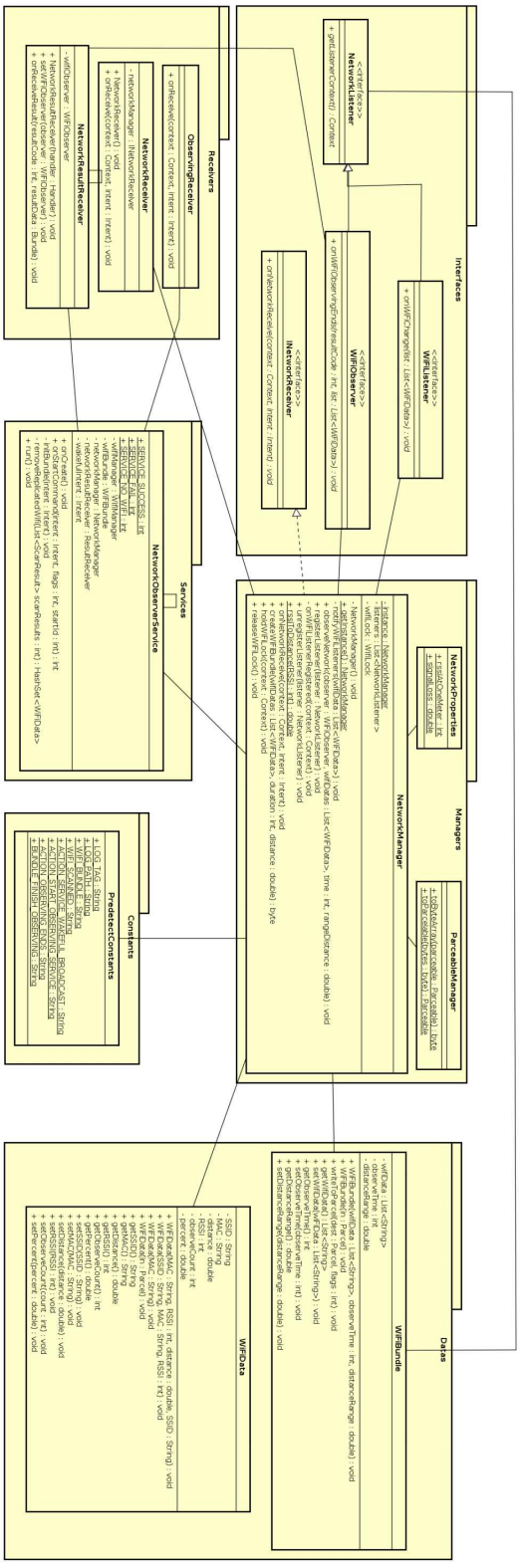
Fonte – Elaborado pelo autor.

APÊNDICE B – DIAGRAMA DE CLASSE DA API PREDETECT MÓDULO WIFI



Fonte – Elaborado pelo autor.

ANEXO A – ANTIGO DIAGRAMA DE CLASSE DA API PREDETECT



Fonte – Filho (2016).