



**UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I
CENTRO CIÊNCIA E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

DOUGLAS RAFAEL OLIVEIRA DOS SANTOS

**DESENVOLVIMENTO DE UM APLICATIVO ANDROID PARA COLETA E
GERENCIAMENTO DE DADOS MÉDICOS DE DISPOSITIVOS COM BLUETOOTH
LOW ENERGY**

**CAMPINA GRANDE - PB
2018**

DOUGLAS RAFAEL OLIVEIRA DOS SANTOS

**DESENVOLVIMENTO DE UM APLICATIVO ANDROID PARA COLETA E
GERENCIAMENTO DE DADOS MÉDICOS DE DISPOSITIVOS COM BLUETOOTH LOW
ENERGY**

Trabalho de Conclusão de Curso em Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de bacharel em Computação.

Área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Paulo Eduardo e Silva Barbosa.

**CAMPINA GRANDE -PB
2018**

É expressamente proibido a comercialização deste documento, tanto na forma impressa como eletrônica. Sua reprodução total ou parcial é permitida exclusivamente para fins acadêmicos e científicos, desde que na reprodução figure a identificação do autor, título, instituição e ano do trabalho.

S237d Santos, Douglas Rafael Oliveira dos.
Desenvolvimento de um aplicativo Android para coleta e gerenciamento de dados médicos de dispositivos com bluetooth low energy [manuscrito] / Douglas Rafael Oliveira dos Santos. - 2018.
65 p. : il. colorido.
Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia, 2018.
"Orientação : Prof. Dr. Paulo Eduardo e Silva Barbosa ,
Coordenação do Curso de Computação - CCT."
1. Prontuário eletrônico. 2. Dispositivo móvel. 3. Aplicativo android. 4. Informação médica. I. Título
21. ed. CDD 004.07

DOUGLAS RAFAEL OLIVEIRA DOS SANTOS

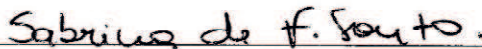
**DESENVOLVIMENTO DE UM APLICATIVO ANDROID
PARA COLETA E GERENCIAMENTO DE DADOS MÉDICOS
DE DISPOSITIVOS COM BLUETOOTH LOW ENERGY
LOW ENERGY**

Trabalho de Conclusão de Curso de Graduação
em Ciência da Computação da Universidade
Estadual da Paraíba, como requisito à
obtenção do título de Bacharel em Ciência da
Computação.

Aprovada em 17 de Agosto de 2018.



Prof. Dr. Paulo Eduardo e Silva Barbosa (DC - UEPB)
Orientador(a)



Profa. Dra. Sabrina de Figueirêdo Souto (DC - UEPB)
Examinador(a)



Profa. Dra. Kézia de Vasconcelos Oliveira Dantas (DC - UEPB)
Examinador(a)

A Deus, pela dedicação, apoio companheirismo e amizade, DEDICO.

AGRADECIMENTOS

A Deus por ter me dado saúde, força, perseverança e sabedoria para concluir este trabalho. A Ele seja dada toda glória, honra e louvor.

A Universidade Estadual da Paraíba por me conceder a oportunidade e privilégio de cursar e concluir o curso de Bacharelado em Ciência da Computação. Agradeço por ter proporcionado infraestrutura necessária e corpo docente.

Ao orientador e Prof. Dr. Paulo Eduardo e Silva Barbosa por ter me proporcionado oportunidades únicas na caminhada acadêmica, como também pela sua confiança, companheirismo e disponibilização das ferramentas necessárias para realização deste trabalho.

Agradeço especialmente ao engenheiro de software Alex Fernandes Figueiredo, pelo acompanhamento em todo o processo de desenvolvimento deste trabalho, dicas e correções. Sem sua ajuda e contribuição talvez não teria alcançado os objetivos traçados.

Aos colegas de curso Edson Sales e Geovannio Vinhas, que se fizeram presentes nos estudos e trabalhos ao longo do curso, proporcionando sempre momentos agradáveis e de qualidade. Ao colega de laboratório Lucas Cosmo e ao amigo de longa data Joanes Miranda, pelo apoio moral nos momentos de escrita. Meus sinceros agradecimentos a toda equipe do laboratório NUTES, que de alguma forma contribuíram para com o trabalho, em especial ao desenvolvedor Fábio Júnior, pelas contribuições realizadas no módulo para visualização dos dados de saúde através de gráficos, tais contribuições foram de grande valor para este trabalho.

Por fim, mas não menos importante, agradeço a toda minha família, minha mãe, Maria do Socorro Oliveira dos Santos, ao meu pai, José Ednaldo dos Santos, minhas irmãs, Fabiana Oliveira dos Santos e Erika Patrícia Oliveira dos Santos e minha avó, Maria José. Sem eles eu não teria a base necessária para concluir essa jornada.

RESUMO

Com os avanços tecnológicos atuais e a afluência da Internet das Coisas, uma série de dispositivos e acessórios vestíveis dedicado ao monitoramento de dados de saúde vêm sendo lançados por grandes fabricantes como Philips, Apple, Nokia, FitBit, Garmin, etc. Estes dispositivos nos permitem realizar um acompanhamento diário da nossa saúde sem a necessidade de ir a um consultório médico ou até mesmo procurar um profissional de saúde, uma vez que dispositivos como termômetro, balança, medidor de pressão arterial, medidor de frequência cardíaca, *smartband* e *smartwatch* passaram a fazer parte do nosso cotidiano. A maioria desses dispositivos usam a tecnologia *Bluetooth Low Energy* (BLE) como principal canal de acesso aos dados coletados, que são transferidos para um smartphone ou tablet. Esta escolha se deve muito pelo fato de o BLE possuir dentre muitos benefícios, o baixo consumo de energia, proporcionando longa autonomia energética para os dispositivos. Em contrapartida, estes dispositivos estão evoluindo sem se preocupar com especificações que garantam a interoperabilidade e conectividade com plataformas desenvolvidas por terceiros, restringindo o seu uso para as soluções proprietárias dos fabricantes e limitando a exploração do potencial que possuem para o domínio da saúde. Tendo isso em vista, neste trabalho de conclusão de curso foi realizado o desenvolvimento de um aplicativo Android para coletar dados de dispositivos pessoais de saúde de diferentes fabricantes por meio do protocolo BLE e integração destes dados na plataforma HANIoT que se encontra em processo de desenvolvimento no Núcleo de Tecnologias Estratégicas em Saúde (NUTES) sediado na Universidade Estadual da Paraíba (UEPB).

Palavras-Chave: Internet das Coisas. Dispositivos Pessoais de Saúde. BLE. Android.

ABSTRACT

With today's technological advancements and the expansion of the Internet of Things, many devices and wearables for monitoring health data have been launched by major manufacturers such as Philips, Apple, Nokia, FitBit, Garmin, etc. These devices allow us to perform daily monitoring of our health without the need to go to a doctor's office or even seek a health professional, since devices such as thermometer, scale, blood pressure meter, heart rate meter, smartband and smartwatch have become part of our daily lives. Most of these devices use Bluetooth Low Energy (BLE) technology as the primary channel for accessing the data collected, which is transferred to a smartphone or tablet. This choice is since the BLE has among many benefits, low energy consumption, providing long energy autonomy for the devices. On the other hand, these devices are evolving without worrying about specifications that guarantee interoperability and connectivity with platforms developed by third parties, restricting their use to manufacturers' proprietary solutions and limiting the exploitation of their potential for health. In this work, the development of an Android application was carried out to collect data from personal health devices from different manufacturers through the BLE protocol and integration of this data into the HANIoT platform that is in the process of development at the Center of Strategic Technologies in Health (NUTES), based at the State University of Paraíba (UEPB).

Keywords: Internet of Things. Personal Health Devices. BLE. Android.

LISTA DE ILUSTRAÇÕES

Figura 1 - Arquitetura do Sistema Android	17
Figura 2 - Arquitetura em camadas da Internet das Coisas	20
Figura 3 - Arquitetura Bluetooth Low Energy	24
Figura 4 - Processo de publicidade entre Central e Periférico.	26
Figura 5 - Processo de comunicação entre Servidor GATT e Cliente GATT	27
Figura 6 - Hierarquia de dados do GATT.....	28
Figura 7 - Arquitetura da plataforma HANIoT	34
Figura 8 - Listagem de Serviços e Características usando o nRF Connect	37
Figura 9 - Operações CRUD/Entidades por segundo utilizando ObjectBox.....	38
Figura 10 - Representação de entidade Measurement utilizando ObjectBox.....	39
Figura 11 - Representação de entidade Contexto utilizando ObjectBox.....	39
Figura 12 - Exemplo de utilização da biblioteca Gson.....	41
Figura 13 - Listagem de algumas <i>User Stories</i> do <i>Product Backlog</i>	43
Figura 14 – Detalhamento da <i>User Story</i> de cadastro de usuário.....	44
Figura 15 - Tela para cadastro de usuário.....	46
Figura 16 - Telas para alteração de dados cadastrais do usuário.....	47
Figura 17 - Tela para autenticação de usuário	48
Figura 18 - Comunicação entre cliente e servidor GATT	49
Figura 19 - Tela do termômetro exibindo medição recebida.....	51
Figura 20 - Telas para visualização do histórico de frequência cardíaca	52
Figura 21 - Telas para visualização do histórico de smartband.....	53
Figura 22 - Tela principal do aplicativo HANIoT.....	54
Figura 23 - Listagem problemas de validação.....	55

LISTA DE TABELAS

Tabela 1 - Comparação entre tecnologias de comunicação sem fio.....	23
Tabela 2 - Propriedades das características	30
Tabela 3 - Informações dos dispositivos BLE selecionados para o trabalho	36

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
CI	Integrated Circuit
CRUD	Create, Read, Update e Delete
DSL	Digital Subscriber Line
GAP	Generic Access Profile
GATT	Generic Attribute Profile
HAL	Hardware abstraction layer
HANIoT	Health ANalytics Internet of Things
HCI	Host Controller Interface
HTTP	Hypertext Transfer Protocol
IoMT	Internet of Medical Things
IoT	Internet Of Things
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
NUTES	Núcleo de Tecnologias Estratégicas em Saúde
OCARIoT	Smart Childhood Obesity Caring Solution using IoT
PHD	Personal Health Device
RFID	Radio-Frequency IDentification
SIG	Special Interest Group
SoC	System on a Chip
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
IBSG	Cisco Internet Business Solutions Group

SUMÁRIO

1 INTRODUÇÃO.....	13
1.1 Motivação	13
1.2 Objetivo	14
1.3 Estrutura do relatório	14
2 FUNDAMENTAÇÃO TEÓRICA	15
2.1 Sistema operacional Android.....	15
2.1.1 Plataforma de código aberto	16
2.1.2 Versões do Android.....	17
2.1.3 Arquitetura.....	17
2.2 Internet das coisas.....	19
2.2.1 Arquitetura da IoT	19
2.2.2 IoT na saúde	21
2.3 <i>Bluetooth low energy</i>	22
2.3.1 Arquitetura do bluetooth low energy.....	23
2.3.2 GAP	24
2.3.3 GATT	26
2.3.3.1 Transações do GATT.....	27
2.3.3.2 Perfis, Serviços e Características.....	27
2.3.3.2.1 Perfis.....	28
2.3.3.2.2 Serviços	29
2.3.3.2.3 Características.....	29
2.4 API REST	31
2.5 Scrum.....	31
2.5.1 Papéis.....	32
3 PLATAFORMA HANIoT.....	33
4 METODOLOGIA.....	35
4.1 Estudo do bluetooth low energy	35
4.2 Estudo dos dispositivos pessoais de saúde	35

4.3 Estudo e seleção de bibliotecas externas	38
4.3.1 ObjectBox	38
4.3.2 OkHttp	40
4.3.3 Gson	40
4.3.4 MPAndroidChart	41
4.4 Processo de desenvolvimento	41
5 RESULTADOS	43
5.1 Especificação	43
5.2 Implementação	45
5.2.1 US01 - Cadastro de usuário	45
5.2.2 US02 - Autenticação de usuário	47
5.2.3 US03 - Cadastro de dispositivos pessoais de saúde	48
5.2.4 US04 - Aquisição de dados dos dispositivos pessoais de saúde	49
5.2.5 US05 - Armazenamento de dados de saúde	51
5.2.6 US06 - Visualização de histórico de dados de saúde	52
5.2.7 US07 - Sincronização de dados de saúde	53
5.3 Validação	55
6 CONCLUSÃO	56
REFERÊNCIAS	58
APÊNDICE A - Product Backlog	61
APÊNDICE B – Parser Heart Rate	65

1 INTRODUÇÃO

Com o avanço da internet e sua popularização, várias áreas da tecnologia da informação têm evoluído, dentre as quais destaca-se o setor de Internet das Coisas (IoT). De acordo com Evans (2011), em 2015 havia 25 bilhões de dispositivos conectados à internet, e o *Cisco Internet Business Solutions Group (IBSG)* prevê que até 2020 haverá 50 bilhões. Dentro do campo da saúde esse avanço tecnológico é conhecido como *Internet of Medical Things (IoMT)*, que se refere a sistemas de dispositivos médicos e aplicações para coleta e integração de dados de saúde por meio de plataformas *online* de tecnologia da informação (ROUSE, 2015).

Impulsionado pelo crescimento do setor de IoT e pela alta demanda do setor de saúde por monitoramento remoto de pacientes, uma série de dispositivos e acessórios vestíveis vêm sendo lançado pelas principais empresas de tecnologia do cenário mundial. Dentre estes encontram-se: termômetros, balanças, medidores de pressão arterial, medidores de frequência cardíaca, pulseiras inteligentes (*smartband*), relógios inteligentes (*smartwatch*), etc.

Grande parte dos dispositivos de IoT usados no monitoramento de variáveis de saúde de pacientes provê comunicação por meio do protocolo *Bluetooth Low Energy (BLE)*, também chamado de *Bluetooth Smart*. O baixo consumo energético é um dos principais fatores para adoção deste protocolo, uma vez que os dispositivos são amplamente usados em cenários de monitoramento contínuo onde o tempo de vida da bateria é um requisito prioritário.

1.1 Motivação

A crescente demanda pela integração de uma grande quantidade de dispositivos de IoT em plataformas de saúde para monitoramento de pacientes resulta em um conjunto de problemas e desafios. Dentre estes, pode-se destacar:

- **heterogeneidade de protocolos** - por mais que se tenha uma especificação mantida pelo *Bluetooth Special Interest Group (Bluetooth SIG)* afim de padronizar os dispositivos que utilizem o protocolo BLE, muitos fabricantes optam por soluções personalizadas, dificultando a integração dos seus dispositivos em plataformas de terceiros. Por exemplo, dois dispositivos BLE de diferentes fabricantes, que não estejam seguindo a especificação oficial do Bluetooth SIG, devem ser integrados a uma plataforma de saúde por meio de soluções distintas que lidam com as especificidades de cada dispositivo, exigindo um maior esforço. Além disso, muitas vezes o fabricante não divulga

nenhuma informação sobre a sua implementação personalizada do protocolo BLE, barrando completamente a integração dos seus dispositivos em plataformas de terceiros;

- **heterogeneidade de aplicações** - a dificuldade em encontrar uma única aplicação que colete e integre dados de dispositivos de diferentes fabricantes. O cenário comum é o que cada fabricante disponibiliza uma solução específica e restrita aos seus dispositivos.

1.2 Objetivo

O objetivo principal deste trabalho é integrar dispositivos pessoais de saúde de diferentes fabricantes à plataforma *Health ANalytics Internet of Things (HANIoT)*, desenvolvida no Laboratório de Engenharia de *Software* do Núcleo de Tecnologias Estratégicas em Saúde (NUTES¹), na Universidade Estadual da Paraíba (UEPB).

Para isso, foi realizado o desenvolvimento de um aplicativo Android para aquisição de dados destes dispositivos por meio do protocolo *Bluetooth Low Energy*, processamento e sincronização destes dados com a plataforma HANIoT, seguindo um modelo de dados comum que garante a interoperabilidade da solução.

O aplicativo desenvolvido no escopo deste trabalho é um protótipo usado para validação da solução proposta, e deve ser evoluído para aplicação em cenários reais de monitoramento de dados de saúde.

1.3 Estrutura do relatório

Este trabalho foi estruturado da seguinte forma: na Seção 2 é apresentada a fundamentação teórica, de forma a proporcionar ao leitor uma referência sobre os principais assuntos abordados no TCC; na Seção 3 é apresentada a plataforma HANIoT, que engloba a aplicação Android desenvolvida no escopo deste trabalho; na Seção 4 é apresentada a metodologia usada para o planejamento e execução do TCC; na Seção 5 é detalhado o processo de desenvolvimento do aplicativo Android, incluindo os resultados obtidos e a validação da solução; e na Seção 6 é apresentada a conclusão do trabalho, onde é realizada todas as considerações finais e são apresentadas sugestões para trabalhos futuros.

¹ <http://nutes.uepb.edu.br>

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, serão apresentados conceitos fundamentais relacionados ao tema para melhor entendimento do contexto em que o projeto está inserido. A seção 2.1 apresenta conceitos sobre o sistema operacional Android, usado para desenvolvimento do aplicativo móvel. A seção 2.2 aborda conceitos de Internet das Coisas (IoT), que é um dos domínios no qual a solução proposta está inserida. A seção 2.3 apresenta o protocolo de comunicação Bluetooth Low Energy, tecnologia sem fio de baixo consumo utilizada pelo aplicativo Android para comunicação e coleta de dados dos dispositivos pessoais de saúde. A Seção 2.4 apresenta o conceito de API REST, padrão usado na comunicação entre o aplicativo Android e o backend para integração dos dados de saúde na plataforma HANIoT. Por fim, a Seção 2.5 apresenta a metodologia de desenvolvimento ágil Scrum, utilizada no gerenciamento e execução das atividades de desenvolvimento do aplicativo Android.

2.1 Sistema operacional Android

De acordo com Tanenbaum (2009), um sistema operacional pode ser elucidado utilizando dois modos distintos. O primeiro modo é uma visão de cima para baixo que consiste em uma abstração do hardware, onde sistema operacional tem a função de apresentar ao usuário o equivalente a uma máquina estendida que é mais fácil de programar e manipular, ou seja, o mediador entre programas e componentes de hardware. Já o segundo modo é uma visão de baixo para cima, onde o sistema operacional é um gerente que controla quais processos podem ser executados, quando e quais recursos (memória, disco, periféricos) podem ser utilizados.

O Android é uma plataforma para dispositivos móveis com tela sensível ao toque como *smartphones* e *tablets*, com também possui versões com interface específica para TVs (*Android TV*), carros (*Android Auto*) e *smartwatch* (*Adroid Wear*). E, de acordo com a StatCounter (2018), empresa que monitora o tráfego na internet, o Android é atualmente o sistema operacional líder absoluto, superando até mesmo o sistema operacional *Windows*.

A plataforma além de possuir a pilha de software denominada sistema operacional, também possui um middleware e aplicações base. Sendo o sistema operacional baseado no kernel do Linux, o qual é responsável pelo gerenciamento de processos, drivers, memória, sistema de arquivos, E/S de rede e energia. O middleware, por sua vez, é o responsável por controlar a interação entre os aplicativos instalados no aparelho, facilitando a comunicação entre eles. Por fim, as aplicações base são os aplicativos como discador, agenda de contatos,

correio de mensagens, navegador, relógio e entre outros. (GLAUBER, 2015, p. 17). Sendo assim, aplicações base estendem as funcionalidades do aparelho e são integradas ao sistema operacional.

A plataforma possui o Android SDK que provê as ferramentas e APIs necessárias para o programador desenvolver aplicativos para o sistema operacional. Atualmente é possível criar aplicações não só para o Android como para outras plataformas como Windows Phone e iOS utilizando a linguagem JavaScript. Entretanto, Java e Kotlin são as linguagens oficiais suportadas para o desenvolvimento nativo de aplicativos para o Android. Sendo o software Android Studio² o principal ambiente para desenvolvimento por conter todas as ferramentas necessárias em um único lugar.

2.1.1 Plataforma de código aberto

O projeto de código do Android é *open source* liderado pela empresa Google e conforme estar licenciado sob licenças comerciais Apache/MIT. Isso permite que fabricantes de aparelhos faça modificações, utilize em seus produtos, e isso não os obriga a compartilhar as modificações com a comunidade de desenvolvimento, nem tão pouco com seus concorrentes. Permitindo assim, que tenham versões do Android personalizadas. (GARGENTA, 2011)

Uma vez que a plataforma é *open source*, isso possibilita que qualquer pessoa possa contribuir com o desenvolvimento, fazendo com que a plataforma seja atualizada com mais frequência e que seja amadurecida mais rápido, ou seja:

“O fato de o Android ser de código aberto contribui muito para seu aperfeiçoamento, uma vez que desenvolvedores de todos os lugares do mundo podem contribuir para seu código-fonte, adicionando novas funcionalidade ou simplesmente corrigindo falhas” (LECHETA, 2009, p. 27).

Entretanto, vale ressaltar que o código do sistema operacional Android não é em sua totalidade livre, e de acordo com Gargenta (2011), existem alguns fragmentos de código de baixo nível que são proprietários, tais como a pilha de código para rede de telefone, wi-fi, bluetooth, GPS, gráficos 3D, câmera, viva voz, entre outros. Ou seja, mesmo fazendo parte do sistema operacional, esses fragmentos de código (firmwares, drivers etc) não são licenciados como código aberto, que permite a legalidade das modificações para fins diversos.

² <https://developer.android.com/studio>

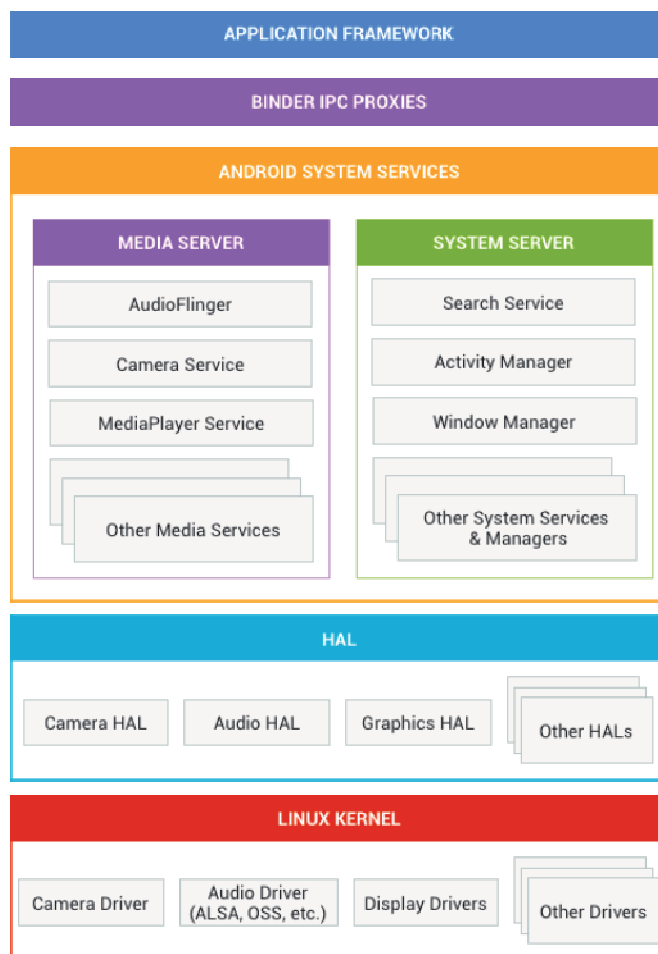
2.1.2 Versões do Android

A primeira versão comercial do Android foi lançada em setembro de 2008, e desde o seu lançamento até os dias de hoje tiveram várias versões, a partir da versão 1.5 a Google atribui ao nome da versão um nome de doce e um número sequencial denominado de *API Level*. (GLAUBER, 2015, p. 18). Neste trabalho o aplicativo foi desenvolvido com suporte a partir da versão 4.4, de nome *KitKat* e *API Level* 19.

2.1.3 Arquitetura

De acordo com Google (2017) a arquitetura do sistema Android contém os seguintes componentes: *Application framework*, *Binder IPC*, *System services*, *Hardware abstraction layer (HAL)* e *Linux kernel*. A Figura 1 ilustra como esses componentes são divididos e organizados.

Figura 1 - Arquitetura do Sistema Android



Fonte: <https://source.android.com/devices/architecture/>

- **Application Framework** - Camada usada com mais frequência pelos desenvolvedores de aplicativos. Como desenvolvedor de hardware, você deve estar ciente das APIs do desenvolvedor, como muitos mapas diretamente para as interfaces HAL subjacentes e pode fornecer informações úteis sobre a implementação de drivers.
- **Binder IPC** - Camada *Binder Inter-Process Communication* permite que a estrutura da aplicação atravesse os limites do processo e invoque o código de serviços do sistema Android. Isso permite que as APIs de estrutura de alto nível interajam com os serviços do sistema Android. No nível da estrutura da aplicação, esta comunicação está escondida do desenvolvedor e as coisas parecem "apenas funcionar".
- **System Services** - Camada que fornece componentes modulares e focados, como *Window Manager*, *Search Service* ou *Notification Manager*. A funcionalidade exposta pelas API da estrutura de aplicativos se comunica com os serviços do sistema para acessar o hardware subjacente. O Android inclui dois grupos de serviços: *System Server* (como *Search Service*, *Window Manager* e *Notification Manager*) e *Media Server* (serviços envolvidos na reprodução e gravação de mídia).
- **Hardware Abstraction Layer (HAL)** - Camada que define uma interface padrão para fornecedores de *hardware*, o que permite que o Android seja agnóstico sobre implementações de *driver* de nível mais baixo. Usar a camada HAL permite que os fabricantes de *hardware* implementem a funcionalidade sem afetar ou modificar o sistema de nível superior. As implementações HAL são empacotadas em módulos e carregadas pelo sistema Android no momento apropriado.
- **Linux kernel** - O Android usa uma versão do kernel do Linux com algumas adições especiais, como *Low Memory Killer* (um sistema de gerenciamento de memória que é mais agressivo na preservação da memória), *Wake Locks* (um serviço de sistema *PowerManager*), o driver *Binder IPC* e outros recursos importantes para uma plataforma móvel. Essas adições são principalmente para funcionalidades do sistema e não afetam o desenvolvimento do driver. Segundo a Google (2017) usar um Kernel do Linux permite que o Android aproveite os recursos de segurança principais e que os fabricantes dos dispositivos desenvolvam drivers de hardware para um kernel conhecido. Entretanto, ao desenvolver um aplicativo o programador nunca interage diretamente com essa camada.

2.2 Internet das coisas

Internet das coisas (do inglês, Internet of Things, IoT) consiste de uma revolução tecnológica com o intuito de conectar à internet objetos usados no dia a dia, tais como, aparelhos domésticos, roupas, relógios, dispositivos médicos, entre outros.

A origem do termo *Internet of Things* foi atribuída a Kevin Ashton que utilizou pela primeira vez como título de uma apresentação que fez na empresa Procter & Gamble em 1999. Ashton definiu internet das coisas como um “sistema capaz de conectar o real e o virtual, criando um mundo mais inteligente em diferentes segmentos da sociedade”. (Apud DIAS, 2016).

De acordo com o Google Trends, desde 2004 a palavra IoT é usada com mais frequência do que *Internet of Things*, seguido de *Web of Things* e *Internet of Everything* como as palavras mais usadas. Citando a mesma referência, Singapura e Índia são os países com maior interesse regional em IoT. Isso muito provável pelo o fato de que a Índia é estimada como o maior consumidor mundial de dispositivos IoT até 2020 (BUYAYA e DASTJERDI, 2016).

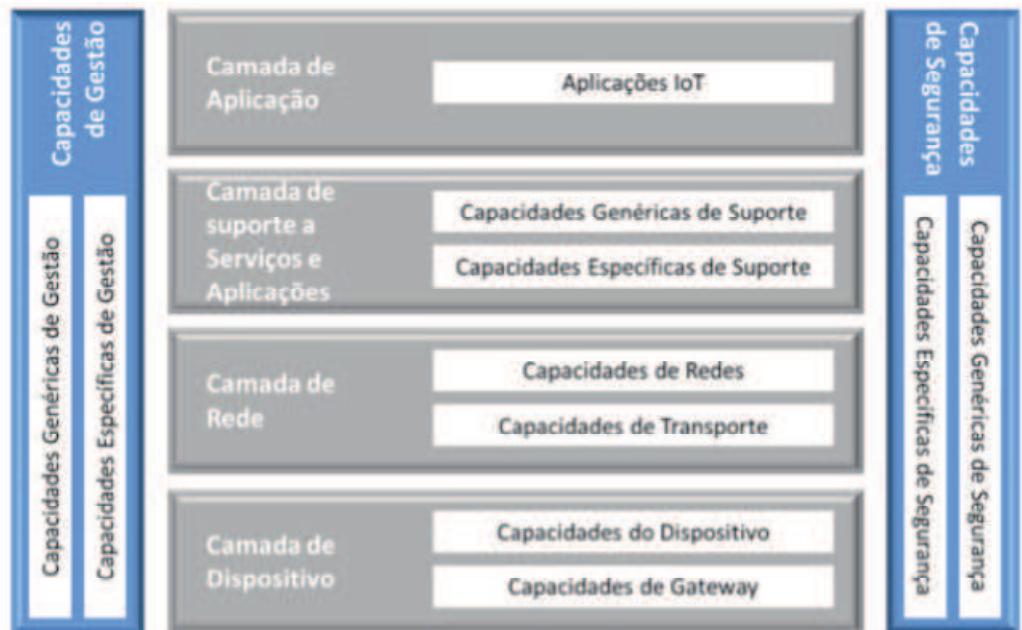
2.2.1 Arquitetura da IoT

De acordo com Filho (2016), a ITU-T definiu a arquitetura da Internet das Coisas em quanto camadas. Sendo elas:

- Camada de aplicação.
- Camada de suporte a serviços de aplicações.
- Camada de rede.
- Camada de dispositivos.

A Figura 2 ilustra o detalhamento das camadas.

Figura 2 - Arquitetura em camadas da Internet das Coisas



Fonte: Recommendation ITU-T Y.2060, 2012, p. 7.

Na **camada de aplicação** temos as aplicações de Internet das Coisas.

A **camada de suporte a serviços de aplicações** de acordo com a ITU-T (2012), possui dois grupos de capacidade de suporte:

- Genéricas – São funcionalidades comuns, que podem ser compartilhadas por diferentes tipos de aplicações IoT, como por exemplo: processamento e armazenamento de dados.
- Específicas – São funcionalidades com atribuições particulares ao domínio, ou seja, específicas para alguma determinada aplicação IoT.

A **camada de rede** é constituída por dois grupos de capacidades, que se referem:

- Às redes de comunicação – São funções para controle de conectividade da rede, tais como, controle das fontes de acesso e transporte, gestão da autenticação, autorização e contabilidade.
- Ao transporte de dados – Prover conectividade para o transporte de dados de aplicações e serviços específicos da IoT.

Por fim, ainda de acordo com a ITU-T (2012) a **camada de dispositivo** é constituída também por dois grupos de capacidades relacionadas aos:

- Dispositivos – Podem ter diversas funcionalidades que incluem interação direta com as redes de comunicação, sem necessidade de gateways. Podem ter interação

indireta, entretanto por meio dos gateways. E ainda, podem ter a capacidade de criar redes particulares para cenários específicos, quando necessário escalabilidade e rápida implantação. Os dispositivos podem ter a capacidade a capacidade de permanecer em estado de descanso, sendo ativados apenas quando se é necessário, isso para conservar energia., tal característica é muito forte no bluetooth low energy, o qual é bastante usado nos dispositivos de IoT.

- Gateways – Suportam diversas interfaces e protocolos, realizando a integração entre os dispositivos e a camada de rede. Podem suportar tecnologias de interface cabeada ou sem fio, como por exemplo: *CANbus*, *ZigBee*, *Bluetooth* e *Wi-Fi*. Como na camada de rede, integrar tecnologias de dados em redes 2G/3G, *Ethernet*, DSL, etc. Os *gateways* devem ter a capacidade de integrar protocolos diferentes, por exemplo: o dispositivo em protocolo *Bluetooth* e a camada de rede em protocolo 3G.

2.2.2 IoT na saúde

A IoT vem revolucionando o mundo e conseqüentemente causando impacto na qualidade de vida das pessoas, isso se dar muito pelos dispositivos comerciais encontrados no mercado atual como: termômetros, balanças, aferidores de pressão arterial, monitores de frequência cardíaca e *wearables* (*smartbands*, *smartwatches*, etc), os quais são vendidos por grandes empresas como Philips, FitBit, Apple e muitas outras.

De acordo com Latam (2018), a Internet das Coisas revolucionará ainda mais nos próximos anos principalmente na área da saúde. Pois, aplicações para saúde de redes IoT seguras podem oferecer cuidado ao paciente em várias frentes, desde ao tratamento para casos agudos em hospitais, até tratamento em longo prazo conhecidos como *Home Care*. O autor ainda enfatiza que apesar de existir uma tendência muito forte de utilização da tecnologia IoT na área de saúde no monitoramento de pacientes, essa tecnologia pode ser bem mais ampla do que isso. Podendo ser as seguintes áreas exploradas:

- Monitoração de pacientes e diagnósticos;
- Transferência de dados, armazenamento e colaboração;
- Dispositivos e ferramentas de saúde inteligentes (cadeira de rodas inteligentes, etiquetas usando RFID, sensores);
- Unidades de emergência conectadas, veículos de resposta e hospitais.

Segundo Cruz (2015) o levantamento feito pela Goldman Sachs mostra que com a IoT o mundo terá uma economia de cerca de 305 bilhões de dólares na saúde com a eliminação de processos custosos nos quais a IoT pode ajudar. Em contrapartida, oportunidades comerciais para empresas que desenvolvam soluções tecnológicas digitais para a saúde chegará a 32 bilhões de dólares por ano, sendo a saúde uma das principais áreas que deverá investir em Internet das Coisas.

2.3 Bluetooth low energy

O *Bluetooth Low Energy* (BLE, também comercializado como Bluetooth Smart) começou como parte da Especificação do Bluetooth 4.0. Foi originalmente projetado pela Nokia como *Wibree* antes de ser adotado pelo *Bluetooth Special Interest Group* (SIG), que é uma organização sem fins lucrativos, a qual supervisiona o desenvolvimento de padrões bluetooth e o licenciamento de dispositivos. A ideia dos os autores não era propor outra solução sem fio excessivamente ampla que tentasse resolver todos os problemas possíveis e sim, projetar um padrão de rádio com o menor consumo de energia, especificamente otimizado para baixo custo, baixa largura de banda, baixa potência e baixa complexidade. Essas metas de design são evidentes por meio da especificação principal, que tenta tornar o BLE um padrão genuíno de baixo consumo de energia. (TOWNSEND, CUFÍ, *et al.*, 2014).

Existe muitos protocolos sem fio disponíveis para fabricantes de produtos IoT, mas o que torna o BLE tão interessante para esse seguimento além das características citadas é, seu suporte a qualquer plataforma móvel existente na atualidade (Android, iOS, Windows Phone, etc). Segue abaixo listagem dos principais sistemas operacionais com as respectivas versões que suportam o BLE.

- iOS5+ (iOS7+ preferido)
- Android 4.3+ (muitas correções de bugs em 4.4+)
- Apple OS X 10.6+
- Windows 8 (XP, Vista e 7 suporta apenas Bluetooth 2.1)
- GNU/Linux Vanilla BlueZ 4.93+

Um dispositivo BLE por ter como característica principal a **economia de energia**, permanece em modo *sleep* durante a maior parte do tempo. Esta tecnologia foi desenvolvida para aplicações que apenas precisam enviar poucas informações, esporadicamente saindo do

modo *sleep* apenas para realizar conexões que duram apenas milissegundos. Com isso, se obtém consumo energético com picos de 6mA, mas com uma média de 1uA apenas. (PESSOA, 2016).

Podemos observar na **Erro! Fonte de referência não encontrada.** uma comparação entre as principais características das tecnologias de comunicação sem fio mais conhecidas na comunicação entre dispositivos: Bluetooth Clássico, Bluetooth Low Energy, ZigBee e Wi-Fi.

Tabela 1 - Comparação entre tecnologias de comunicação sem fio

	Bluetooth Clássico	Bluetooth Low Energy	ZigBee	Wi-Fi
Padrão de camada física	IEEE 802.15.1	GFSK	IEEE 802.15.4	IEEE 802.11abg
Frequências	2.4 GHz	2.4 GHz	868 MHz, 915 MHz, 2.4 GHz	2.4 GHz, 5 GHz
Máxima Taxa de Bits (Mbps)	1 até 3	1	0.25	11(b), 54(g), 600(n)
Distância Máxima (m)	10-100	50	10-100	100-250
Consumo de Energia	Alto	Muito Baixo	Muito Baixo	Muito alto
Vida Útil da Bateria	Dias	Meses a anos	Meses a anos	Horas
Tamanho da Rede	7 dispositivos	Indefinido	64,000+ dispositivos	255 dispositivos

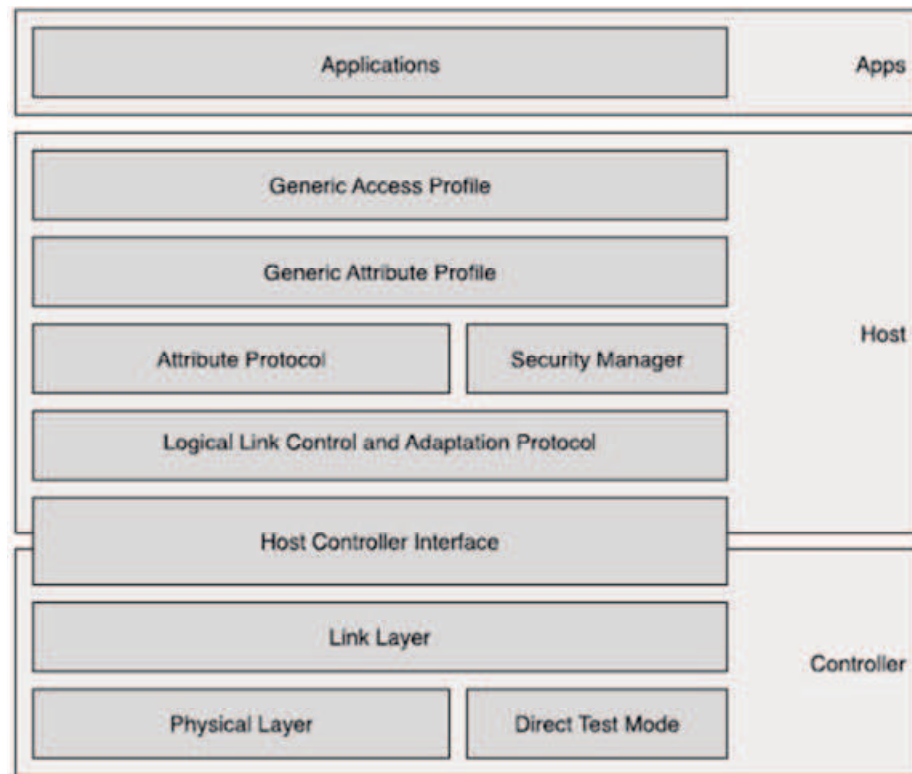
Fonte: PESSOA (2016)

2.3.1 Arquitetura do bluetooth low energy

De acordo com Pessoa (2016) a arquitetura do BLE é dividida em duas camadas, sendo elas *Host* e *Controller* Figura 3. Sendo assim, os dispositivos podem ser produzidos por fabricantes distintos, se comunicando através de uma camada *Host Controller Interface* (HCI), o que permite a interoperabilidade entre os diferentes fabricantes. Pessoa (2016) ainda enfatiza que o fabricante pode implementar um *System on a Chip* (SoC) que consiste em um único

Integrated Circuit (CI) com o *host* e *controller*, o que reduz os custos de produção. O *controller* é o responsável por permitir o *host* permanecer no modo *sleep* por longos períodos, despertando somente quando se é necessário realizar alguma ação.

Figura 3 - Arquitetura Bluetooth Low Energy



Fonte: HEYDON (2012)

2.3.2 GAP

O *Generic Access Profile* (GAP) é o responsável direto por controlar as conexões e publicidade, permite que dispositivos BLE interajam entre si. Ele fornece uma estrutura que qualquer dispositivo com Bluetooth Low Energy deve seguir para permitir que os dispositivos se descubram, transmitam dados, estabeleçam conexões seguras e executem muitas outras operações fundamentais seguindo o padrão. (TOWNSEND, CUFÍ, *et al.*, 2014).

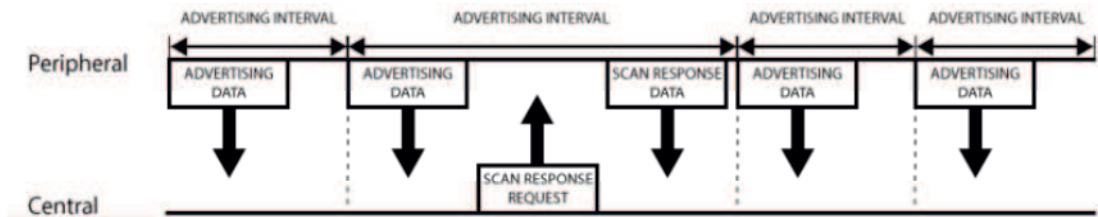
Segundo Pessoa (2016) o GAP fornece um *framework* que qualquer implementação BLE deve seguir para que dispositivos descubram outros e sejam descobertos, como também, estabeleçam conexões seguras e forneçam dados de *broadcast*.

O GAP define diferentes aspectos da interação dos dispositivos, são eles, *Roles*, *Modes*, *Procedures* e *Security*:

- **Roles** (Papéis) – Cada dispositivo pode operar uma ou mais funções simultaneamente. Cada papel estabelece restrições e impõe certos requisitos comportamentais. Certas combinações de funções permitem que os dispositivos se comuniquem entre si, e o GAT estabelece as iterações entre esses papéis com precisão. De acordo com Pessoa (2016) os papéis são divididos em 4 especificações diferentes: *broadcaster*, *observer*, *central* e *Peripheral*. Na especificação *broadcaster*, os dados são transmitidos através dos *advertising channels* sem poder se conectar a outros dispositivos. Na *observer*, são os dispositivos que buscam anunciantes, sem poder iniciar uma conexão. Já na *central*, dispositivos buscam anunciantes e iniciam uma conexão, atuando assim como mestres. E, em *peripheral* um anunciante se conecta, atuando como escravo;
- **Modes** (Modos) – Estado que pode conter variações de acordo como o tempo necessário para alcançar um objetivo, ou seja, permitir um ponto realizar um processo específico. (PESSOA, 2016);
- **Procedures** (Procedimentos) – Sequência de ações que permite o dispositivo atingir um determinado objetivo. Um procedimento estar diretamente associado a um modo de outro par, então, na maioria das vezes o modo e procedimento são fortemente acoplados.
- **Security** (Segurança) – Define modos de segurança e procedimentos, os quais especificam como os pares estabelecem o nível de segurança requerido em uma troca de dados particular.

Existem duas maneiras de enviar publicidade com o GAP, uma através de *Advertising Data* (Dados de Publicidade) e outra por *Scan Response* (Resposta de Escaneamento). Segundo Townsend (2015), ambos payloads são idênticos e podem conter até 31 bytes de dados, porém apenas *Advertising Data* é obrigatória, pois será constantemente transmitida do dispositivo para permitir que outros dispositivos centrais saibam de sua existência. Já o payload *Scan Response* é uma carga útil opcional que as centrais podem solicitar, isso permite que os designers de dispositivos ajustem um pouco mais de informações na carga útil como cadeias de caracteres para um nome de dispositivo. A Figura 4 ilustra o processo de publicidade e como funcionam os payloads *Advertising Data* e *Scan Response*.

Figura 4 - Processo de publicidade entre Central e Periférico.



Fonte: (TOWNSEND, 2015)

Um periférico define um intervalo de publicidade específico e, toda vez que esse intervalo passar, ele retransmitirá seu principal pacote de publicidade. Se um dispositivo de escuta estiver interessado no *Scan Response*, *ele* poderá solicitar opcionalmente a carga útil da resposta, e o periférico responderá com os dados adicionais. (TOWNSEND, 2015).

2.3.3 GATT

O Generic Attribute Profile (GATT) define o modo como dois dispositivos BLE devem transmitir dados, em contraste com o GAP, que define as interações de baixo nível com os dispositivos, o GATT fica responsável em lidar apenas com procedimentos e formatos reais de transferência de dados. Para isso, é utilizado conceitos chamados de *Services* (Serviços) e *Characteristics* (Características). Ele faz uso do protocolo de dados genérico chamado de *Attribute Protocol* (ATT), o qual é usado para armazenar os serviços, características e dados relacionados em uma tabela de consulta simples usando UUIDs, um número de 16 bytes que é garantido (ou tem alta probabilidade) de ser globalmente exclusivo. (TOWNSEND, CUFÍ, *et al.*, 2014, p. 51, 52).

O GATT entra ação quando uma conexão dedicada é estabelecida entre dois dispositivos, isso implica que o processo de publicidade regido pelo GAP já foi passado. É importante ressaltar que as conexões são exclusivas. O seja:

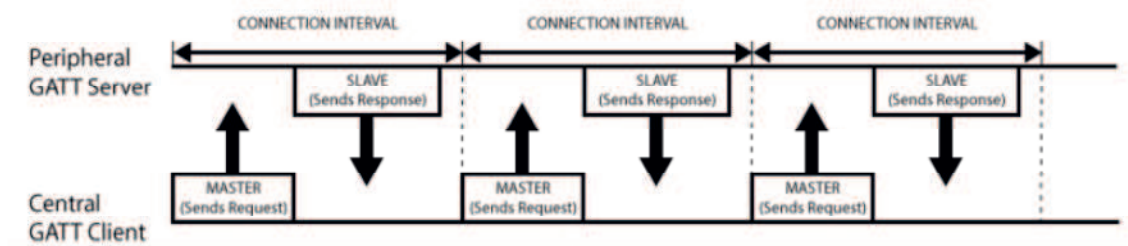
“[...] um periférico BLE só pode ser conectado a um dispositivo central (um celular, etc.) por vez! Assim que um periférico se conecta a um dispositivo central, ele interrompe a publicidade e outros dispositivos não poderão mais vê-lo ou conectá-lo até que a conexão existente seja interrompida. Estabelecer uma conexão também é a única maneira de permitir a comunicação bidirecional, onde o dispositivo central pode enviar dados significativos para o periférico e vice-versa.” (TOWNSEND, 2015, tradução nossa).

2.3.3.1 Transações do GATT

Um conceito muito importante no GATT é o relacionamento entre servidor/cliente. O periférico é conhecido como **servidor GATT**, o qual contém os dados de consulta ATT, definições de serviços e características. Já o **cliente GATT** (telefone, tablet, etc) é o que envia solicitações para o servidor. (TOWNSEND, 2015). Ou seja, todas as transações são iniciadas pelo dispositivo mestre, conhecido como o cliente GATT, que recebe respostas do dispositivo escravo, que é o servidor GATT.

O diagrama representado pela Figura 5 ilustra o processo de comunicação (troca de dados) entre um servidor (o periférico) e um cliente (o dispositivo), com o dispositivo mestre iniciando cada transação.

Figura 5 - Processo de comunicação entre Servidor GATT e Cliente GATT



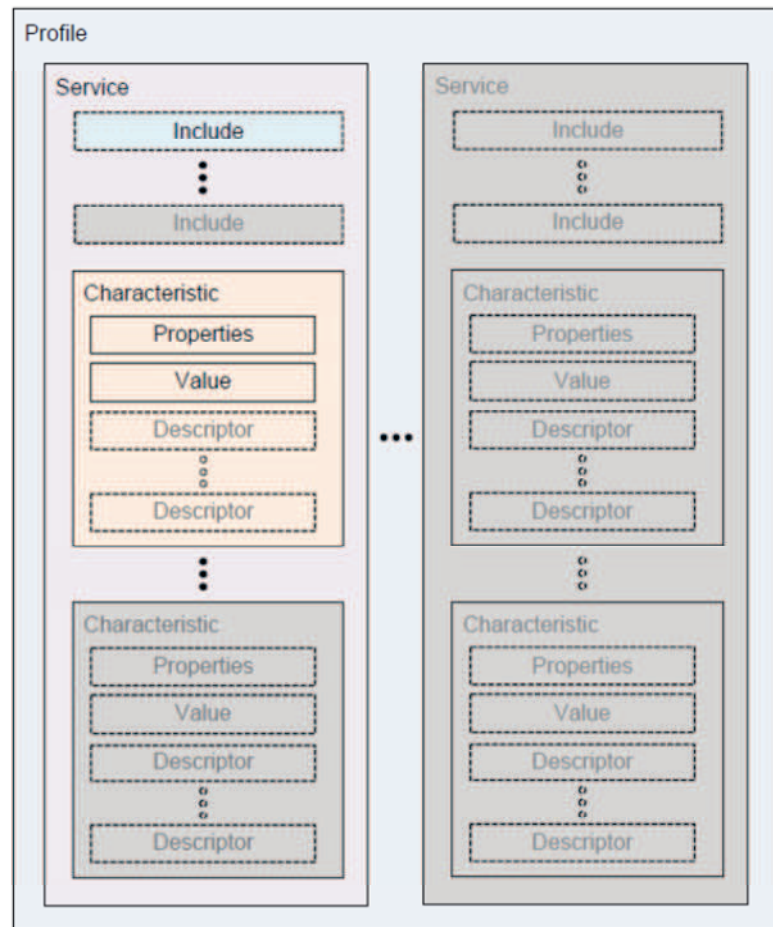
Fonte: (TOWNSEND, 2015)

Ao estabelecer uma conexão, o periférico sugerirá um intervalo de conexão para o dispositivo central, o qual tentará reconectar a cada intervalo de conexão para ver se existem novos dados disponíveis.

2.3.3.2 Perfis, Serviços e Características

As transações do GATT são baseadas em objetos aninhados de alto nível chamados de: *Profiles*, *Services* e *Characteristics*, conforme a Figura 6.

Figura 6 - Hierarquia de dados do GATT



Fonte: (BHARGAVA, 2017).

2.3.3.2.1 Perfis

Um perfil em si, não existe no próprio periférico BLE, é apenas uma coleção pré-definida de serviços que foi estabelecida e compilada pelo Bluetooth SIG³, ou pelos designers de periféricos. Por exemplo, o perfil do periférico termômetro combina o serviço de temperatura e o serviço de informações do dispositivo.

Como mostra na a Figura 6, o perfil dita quais serviços cada perfil hospeda, ou seja, define a finalidade do periférico estabelecendo o que ele pode fazer. (BHARGAVA, 2017).

³ <https://www.bluetooth.com>

2.3.3.2.2 Serviços

Os serviços são usados para dividir dados em entidades lógicas contendo partes específicas de dados chamadas de características. Podendo o serviço ter uma ou mais características. Cada serviço é discriminado de outros serviços por meio de um UUID exclusivo, que pode ser de 16 bits para serviços definidos pelo SIG⁴ ou 128 bits para os serviços personalizados. Vale ressaltar que o fabricante que opta em criar serviços personalizados em seus dispositivos dificulta a possibilidade de desenvolvedores consumirem os dados do periférico via bluetooth Low Energy, pois apenas o fabricante tem conhecimento de quais características e como foram implementadas nos serviços personalizados, dificultando assim a interoperabilidade.

Conceitualmente, um serviço GATT pode ser comparado com uma classe em qualquer linguagem orientada a objeto moderna, completa com instanciação, porque um serviço pode ser instanciado várias vezes em um único servidor GATT (no entanto, isso não é uma ocorrência comum e na maioria dos serviços a instância é única, semelhante aos singletons). (TOWNSEND, CUFÍ, *et al.*, 2014, p. 58)

2.3.3.2.3 Características

As características é o nível mais baixo nas transações do GATT. São encapsuladas por um serviço relacionado, ou seja, estão sempre associadas a um serviço, e cada característica contém um único ponto de dados, embora possa conter uma matriz de dados relacionados, como valores X/Y/Z de um acelerômetro de 3 eixos. Além disso, o valor da característica pode ser seguido por descritores, os quais expandem ainda mais os metadados contidos na declaração das características.

Segundo Townsend (2014), de forma similar aos serviços cada característica se distingue de outras por meio de um UUID único de 16 bits para as características que estão definidas no SIG⁵ (que garante a interoperabilidade entre hardware e software habilitados com BLE) ou 128 bits para as características personalizadas, as quais somente o periférico e software do próprio fabricante entendem.

As propriedades de uma característica são agrupadas em 8 bits, juntamente com os dois bits adicionais no descritor de propriedades estendidas, determinam como o valor da

⁴ <https://www.bluetooth.com/specifications/gatt/services>

⁵ <https://www.bluetooth.com/specifications/gatt/characteristics>

característica pode ser usado e como os descritores podem ser acessados. A Tabela 2 contém os tipos de propriedades que podem ser usados em uma característica.

Tabela 2 - Propriedades das características

Propriedade	Localização	Descrição
Broadcast	Properties	Se definido, significa que o valor desta característica será transmitido, isto é, colocado nos pacotes de publicidade.
Read	Properties	Se definido, significa que os clientes podem ler o valor dessa característica.
Write without response	Properties	Se definido, significa que os clientes podem gravar (sem resposta) o valor dessa característica.
Write	Properties	Se definido, significa que os clientes podem gravar (e receber uma resposta) ao valor dessa característica.
Notify	Properties	Um dos mais importantes. Se definido, o servidor notificará assincronamente o cliente sempre que o valor da característica for atualizado no servidor. Além disso, se definido, o descritor de configuração do cliente existirá.
Indicate	Properties	Similar a notificar, a única diferença é que uma indicação requer uma confirmação do cliente. Além disso, se definido, o descritor de configuração do cliente existirá.
Signed Write Command	Properties	Se definido, significa que os clientes podem fazer uma gravação assinada no valor dessa característica.
Queued Write	Extended Properties	Se definido, significa que o cliente pode enfileirar gravações nessa característica.
Writable Auxiliaries	Extended Properties	Se definido, o cliente pode gravar no descritor de descrição do usuário da característica.

Fonte: (BHARGAVA, 2017).

2.4 API REST

O acrônimo API provém do inglês *Application Programming Interface*, que em português significa, Interface de Programação de Aplicações. Trata-se de um conjunto de rotinas, padrões estabelecidos e documentados por uma determinada aplicação, para que outras aplicações consigam utilizar as suas funcionalidades sem precisar conhecer detalhes da implementação do software. (PIRES, 2017). Sendo assim, por meio das APIs é possível a integração entre softwares implementados em linguagens diferentes de maneira ágil e segura.

REST significa *REpresentational State Transfer* (Transferência de Estado Representativo). É um estilo de desenvolvimento de serviços web, que dentre outros preceitos é guiado pelas boas práticas de uso do protocolo *HyperText Transfer Protocol* (HTTP), como o uso adequado dos: métodos e URL's, código de status padronizados para representação de falha e sucesso, cabeçalhos e interligações entre vários recursos diferentes. (SAUDATE, 2014).

2.5 Scrum

Framework para desenvolver e manter produtos complexos tendo um modelo estrutural que está sendo usado para gerenciar o desenvolvimento de produtos complexos desde o início de 1990. Scrum não é um processo nem mesmo uma técnica para desenvolver produtos, é um framework dentro do qual é possível empregar vários processos ou técnicas. O Scrum deixa claro a eficácia relativa as práticas de gerenciamento e desenvolvimento de produtos, de modo que você possa melhorá-las. (SCHWABER e SUTHERLAND, 2013, p. 3).

Segundo Sutherland (2014, p. 16) o termo Scrum vem do jogo rúgbi e se refere a maneira como uma equipe trabalha junta avançando com a bola no campo. Alinhamento cauteloso, unificação de propósito, clareza de objetivo e tudo unido.

No Scrum o desenvolvimento do projeto é dividido em ciclos (semanal, mensal, etc) que são chamados de *Sprints*. As funcionalidades a serem implementadas são descritas em uma lista chamada de *Product Backlog*, e cada funcionalidade é chamada de *User Story* que contém uma descrição de uma necessidade do cliente de forma simples. No início de cada Sprint é realizada uma reunião de planejamento na qual o *Product Owner* prioriza os itens do *Product Backlog* e juntamente com a equipe seleciona as atividades que serão implementadas durante o *Sprint* que se inicia.

2.5.1 Papéis

- **Equipe** – Reesposáveis pela a entrega das tarefas, normalmente é formada por grupo pequeno entre 5 e 9 pessoas, que trabalham de forma auto gerenciada. É a própria equipe que fica responsável em estimar o tempo para concluir cada tarefa, garantir a qualidade na entrega e apresentar ao cliente o produto a cada fim de *sprint*;
- **Product Owner** – É a pessoa responsável pela visão de negócios do projeto. É ele que prioriza o *Product Backlog* e na maioria das vezes é o próprio cliente que desempenha esse papel;
- **Scrum Master** – Integrante da equipe que tem como principal função ser um facilitador e mediador entre o cliente e desenvolvedor. Seu papel é assegurar que as práticas de Scrum estão sendo aplicados com eficiência.

3 PLATAFORMA HANIoT

A plataforma HANIoT (*Health ANalytics Internet of Things*) é desenvolvida e mantida pela equipe de desenvolvimento do laboratório de engenharia de software do Núcleo de Tecnologias Estratégicas em Saúde (NUTES) com sede na Universidade Estadual da Paraíba (UEPB). O desenvolvimento da plataforma é realizado em parceria com pesquisadores e profissionais da saúde como: endocrinologistas, nutricionistas, fisioterapeutas, entre outros.

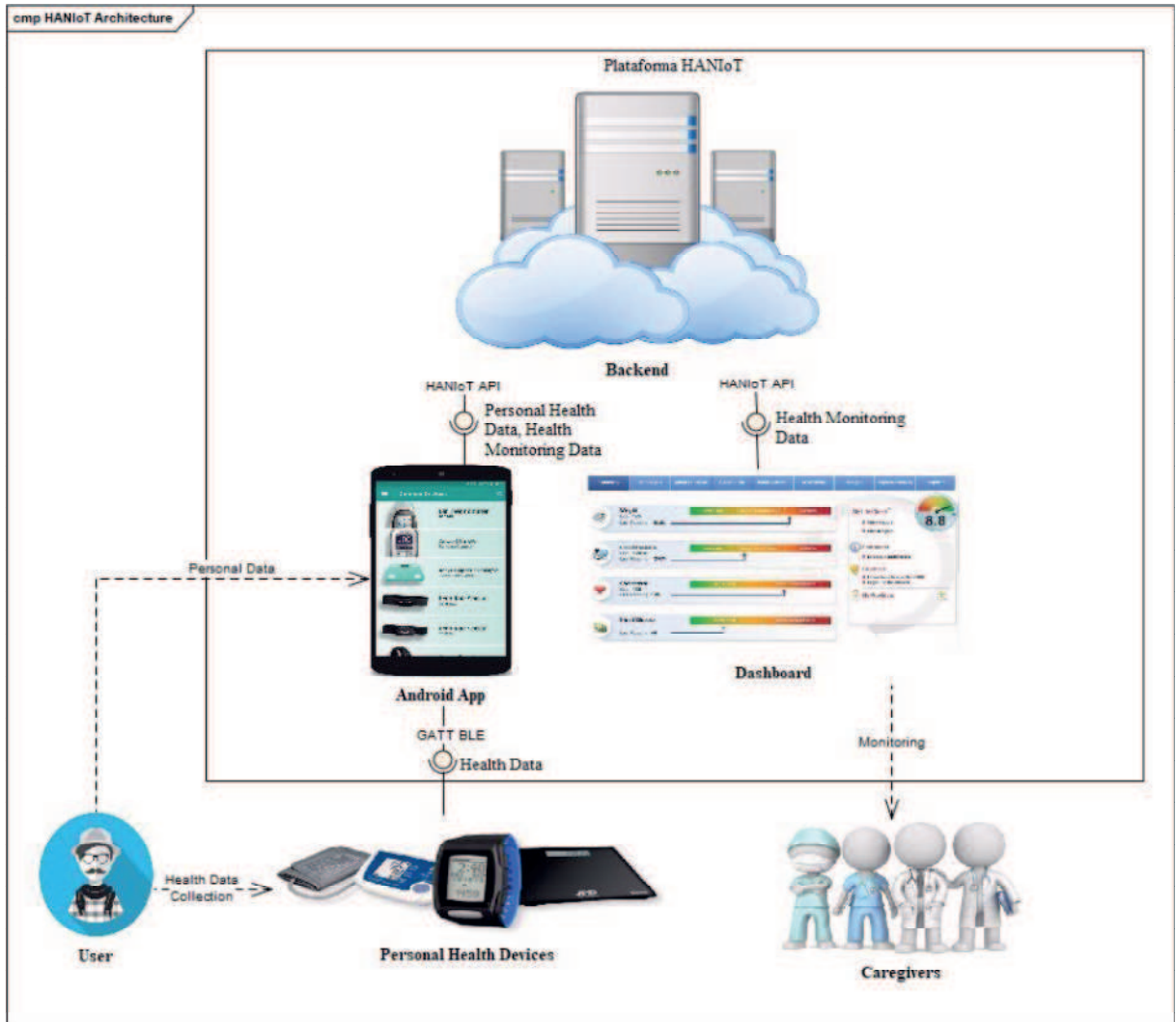
Na Figura 7 é apresentada a arquitetura da plataforma HANIoT, que basicamente é composta por três componentes: *Backend*, Aplicativo Android e *Dashboard* (aplicação Web). A arquitetura da plataforma segue o padrão Cliente-Servidor, sendo o *Backend* o componente servidor e as aplicações Android e Web os componentes cliente da solução.

O componente servidor foi desenvolvido em (NETO, 2018), sendo o responsável por receber dados pessoais de saúde do aplicativo Android e disponibiliza-los para monitoramento por meio de ambas as aplicações Android e Web. O componente oferece uma API Rest para comunicação com as aplicações cliente da plataforma, provendo dados de forma uniformizada e abstraindo os padrões personalizados de cada fabricante, garantindo assim, a interoperabilidade da solução no que diz respeito a integração de dados de diferentes dispositivos.

A aplicação cliente Android é o componente desenvolvido no escopo deste trabalho, com o objetivo de realizar a aquisição dos dados de dispositivos pessoais de saúde e integrá-los a solução, seguindo os modelos de dados definidos para a plataforma HANIoT. A comunicação com os dispositivos pessoais de saúde é feita por meio do protocolo BLE.

A aplicação web é o componente da solução usado pelos profissionais de saúde para o gerenciamento e monitoramento dos dados médicos coletados pela plataforma, além disso, a aplicação agrega conjuntos de funcionalidades avançadas para facilitar a visualização e análise dos dados coletados. Esse componente encontra-se em processo de desenvolvimento pela equipe desenvolvimento HANIoT.

Figura 7 - Arquitetura da plataforma HANIoT



Fonte: Próprio autor

4 METODOLOGIA

Nesta seção serão apresentadas as principais etapas para realização deste trabalho. A pesquisa pode ser classificada como exploratória, pois antes do processo de desenvolvimento do aplicativo Android, foi necessário o estudo detalhado do protocolo de comunicação BLE, dos dispositivos pessoais de saúde do mercado (termômetro, balança, medidor de pressão arterial, *smartband*, *smartwatch*, etc) e estudo das bibliotecas Android utilizadas para a implementação do aplicativo, como também, o levantamento dos requisitos.

4.1 Estudo do bluetooth low energy

Nesta etapa foi realizado o estudo detalhado do protocolo de comunicação BLE, usado para aquisição de dados dos dispositivos pessoais de saúde. Neste estudo foi utilizada a documentação oficial disponível no site do Bluetooth SIG⁶ para entender a especificação GATT (serviços, características, etc.) usada na troca de dados por meio do protocolo BLE. Após o entendimento do protocolo, foi realizada pesquisas por bibliotecas BLE que pudessem ser utilizadas no desenvolvimento do aplicativo Android. Diante das investigações, optou-se por usar o próprio SDK do Android que disponibiliza uma biblioteca BLE⁷ com todos os recursos necessários para comunicação com os dispositivos pessoais de saúde.

4.2 Estudo dos dispositivos pessoais de saúde

Nesta etapa foram realizadas investigações em manuais de dispositivos comerciais disponíveis no Laboratório de Engenharia de Software do NUTES, assim como a consulta a alguns fabricantes, de modo a obter informações gerais e a especificação do protocolo BLE (serviços e características) de cada dispositivo, necessário para a integração destes com aplicativo Android. Na Tabela 3 são listados os dispositivos selecionados e algumas informações coletadas, por exemplo: nome do dispositivo, tipo de medições, acurácia, alcance, duração de bateria e versão do protocolo BLE.

⁶ <https://www.bluetooth.com>

⁷ <https://developer.android.com/guide/topics/connectivity/bluetooth-le>

Tabela 3 - Informações dos dispositivos BLE selecionados para o trabalho

Dispositivo	Tipo de Medição	Acurácia	Alcance	Bateria	Versão BLE
Philips Ear Thermometer	Temperatura	±0.2°C	32.4°-42.9°C	±1 ano	4.0
Accu-Chek® Performa Connect	Glicose no sangue	±10 mg/dL	20-600mg/dL	750 leituras	4.0
YUNMAI Mini 1501	Peso, Gordura Corporal, Gordura Visceral, Massa Óssea, BMR ⁸ , BMI ⁹ , Taxa Muscular, Água no Corpo, Idade Fitness e Proteínas	Peso: ±0.1Kg Gordura C.: 0.1% Gordura V.: 0.1% Massa Óssea: 0.1Kg BMI: 0.1% Taxa Muscular: Água no C.: 0.1% Idade F.: 1-3 anos Proteínas: 0.1%	Peso: 3-180Kg	-----	4.0
Polar H10 Heart Rate Sensor	Frequência Cardíaca, Calorias Queimadas	Alta precisão	----	400 horas	4.0
Xiaomi Mi Band 2	Passos, Quilômetros Percorridos, Calorias Queimadas, Sono e Frequência Cardíaca	----	----	+20 dias	4.0

Fonte: Próprio autor

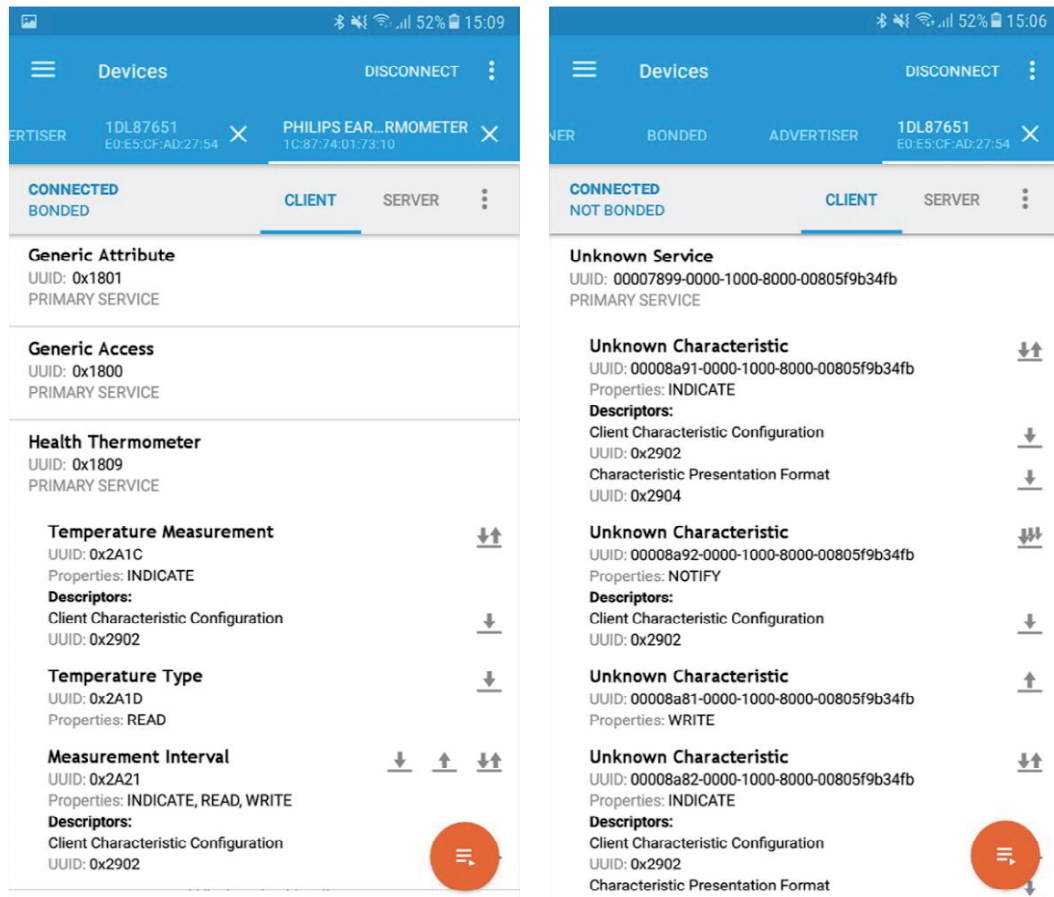
O aplicativo desenvolvido pela empresa Nordic Semiconductor, chamado de nRF Connect for Mobile (disponível na loja de aplicativos do Android¹⁰), foi utilizado na investigação da especificação GATT de cada dispositivo. Através do aplicativo é possível realizar escaneamento dos dispositivos BLE próximos, estabelecer a conexão com o dispositivo desejado, e ter acesso a informações detalhadas dos serviços e características implementados por este, como pode ser observado na Figura 8 (à esquerda). Sendo assim, foi possível identificar não apenas os tipos de serviços implementados e suas características, mas também suas propriedades (Leitura, Escrita, Notificação, Indicação, etc).

⁸ Basal Metabolic Rate (Taxa Metabólica Basal)

⁹ Body Mass Index (Índice de Massa Corporal)

¹⁰ <https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp>

Figura 8 - Listagem de Serviços e Características usando o nRF Connect



Fonte: Próprio autor

Como citado na Seção 2.3.3.2.2, o fabricante pode optar por implementar o protocolo BLE de forma personalizada, em vez de seguir a especificação do Bluetooth SIG. Isso pode ser observado na Figura 8 (à direita), onde os serviços com a descrição “*Unknown Service*” (Serviço Desconhecido) são serviços personalizados, não descritos na documentação oficial do GATT. Essa abordagem, adotada por boa parte dos fabricantes de dispositivos pessoais de saúde, dificulta ou impossibilita a integração destes dispositivos em plataformas de terceiros por meio do protocolo BLE.

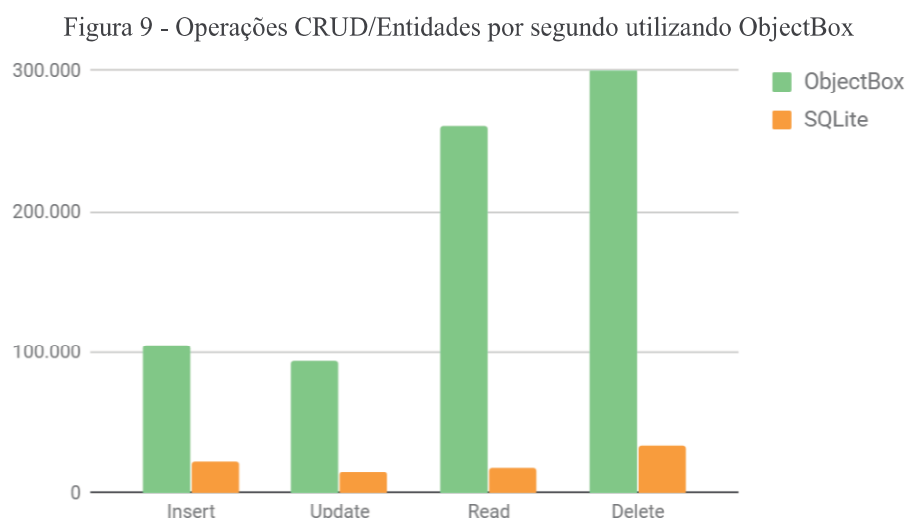
Dentre os dispositivos identificados com implementação personalizada do protocolo GATT, temos o aferidor de pressão sanguínea DL8760 da Philips e o Smartwatch FitBit Ionic. Pelo fato de não seguirem a especificação oficial do Bluetooth SIG e não se ter acesso a documentação especificando a implementação personalizada usada pelos fabricantes, não foi possível integra-los ao aplicativo Android desenvolvido.

4.3 Estudo e seleção de bibliotecas externas

Nesta etapa foi realizada uma pesquisa por bibliotecas Android que poderiam ser utilizadas para o desenvolvimento do aplicativo, com a finalidade de diminuir o tempo necessário para sua implementação. A pesquisa foi centralizada em bibliotecas para persistência de dados, requisições HTTP, transformações entre JSON e Objetos, plotagem de gráficos, etc. As principais bibliotecas selecionadas e utilizadas no desenvolvimento são listadas a seguir.

4.3.1 ObjectBox

Biblioteca para persistência de dados com uso de armazenamento por chave-valor em vez do armazenamento com colunas (bancos de dados relacionais). Essa abordagem resulta em um desempenho 10x mais rápido o que o banco de dados SQLite (banco de dados oficial para armazenamento no Android). O ObjectBox é extremamente rápido, e isso independe da quantidade de dados ou do dispositivo que você está usando. A Figura 9 apresenta a comparação entre o ObjectBox e o SQLite em operações: *Create*, *Read*, *Update* e *Delete* (CRUD). (OBJECTBOX, 2018). Sua escolha se deu principalmente pela facilidade de uso e rapidez nas operações de persistência.



Fonte: (OBJECTBOX, 2018)

Usando anotações como `@Entity` no modelo de dados definido, sinalizamos que o objeto deverá ser persistido no banco e podemos definir relacionamentos entre entidades de

maneira muito simples. Nas figuras 9 e 10 podemos observar a definição das entidades Medição e Contexto, respectivamente, utilizadas no desenvolvimento da solução.

Figura 10 - Representação de entidade Measurement utilizando ObjectBox

```

1  @Entity
2  public class Measurement {
3      @Id private long id;
4      private double value;
5      private String unit;
6      private long registrationDate;
7
8      private ToOne<User> user;
9      private ToOne<Device> device;
10     private ToMany<ContextMeasurement> contextMeasurements;
11     private ToMany<Measurement> measurements;
12     private int typeId;
13
14     public Measurement() {
15     }
16 }

```

Fonte: Próprio autor

Figura 11 - Representação de entidade Contexto utilizando ObjectBox

```

1  @Entity
2  public class ContextMeasurement {
3      @Id private long id;
4      private int valueId;
5      private int typeId;
6
7      private ToOne<Measurement> measurement;
8
9      public ContextMeasurement() {
10     }
11 }

```

Fonte: Próprio autor

A multiplicidade dos relacionamentos entre as entidades do banco é especificada como descrito a seguir:

- **relacionamento um para um (1..1)** – usa-se a especificação *ToOne<Object>* na entidade de origem. Como por exemplo, na Figura 9, onde a classe *Measurement* possui um relacionamento *ToOne<User>*;
- **relacionamento um para muitos (1..n)** – usa-se a especificação *ToMany<Object>* na entidade de origem e *ToOne<Object>* na entidade de destino. Como por exemplo, nas Figura 10 e Figura 11, onde a classe *Measurement* possui *ToMany<ContextMeasurement>*, e a classe *ContextMeasurement* possui *ToOne<Measurement>*;

- **relacionamento muitos para muitos (n..n)** – usa-se apenas a especificação *ToMany<Object>* na entidade de origem. Como por exemplo, na Figura 10, onde a classe *Measurement* possui *ToMany<Measurement>*.

4.3.2 OkHttp

Biblioteca para realização de requisições HTTP de fácil utilização. É um projeto de código livre¹¹. Suporta o protocolo SPDY (base do HTTP 2.0) e permite que várias solicitações sejam multiplexadas em um mesmo soquete. O OkHttp persevera quando a rede é problemática, ou seja, ela se recupera silenciosamente de problemas comuns relacionados a conexão. (SQUARE, 2018).

Na implementação desse trabalho foi desenvolvido um módulo específico para comunicação com servidor externo e a biblioteca OkHttp é a interface principal para funcionamento desse módulo. Na seção 5.2.75.2. é exposto mais detalhes dessa implementação.

4.3.3 Gson

Biblioteca de código livre¹² para converter objetos Java em JSON e fazer o processo inverso convertendo uma String JSON em um objeto Java equivalente. Com o Gson é possível trabalhar com objetos Java arbitrários, incluindo objetos pré-existentes que não se tem o código fonte. Essa biblioteca foi bastante utilizada neste trabalho antes de enviar qualquer dado para o servidor, convertendo os objetos Java em JSON, e quando se recebia do servidor uma String no formato JSON, convertendo para o modelo de dado Java correspondente. Na Figura 12 podemos ver um exemplo desses dois processos.

¹¹ <https://github.com/square/okhttp>

¹² <https://github.com/google/gson>

Figura 12 - Exemplo de utilização da biblioteca Gson

```
1  Gson gson = new Gson();
2
3  User user = new User();
4  user.setName("João da Silva");
5  user.setEmail("joao@mail.com");
6
7  // Convertendo Objeto em String JSON
8  String userJSONString = gson.toJson(user);
9
10 // Convertendo String JSON em Objeto
11 User newUser = gson.fromJson(userJSONString, User.class);
```

Fonte: Próprio autor

4.3.4 MPAndroidChart

Biblioteca de código livre¹³ selecionada para geração de gráficos no aplicativo Android. Apesar de ser uma biblioteca relativamente complexa, foi identificado que ela é bastante utilizada pela comunidade de desenvolvedores Android devido a sua maturidade, abrangência de funcionalidades, boa documentação e possibilidades de personalização em níveis básico e avançado.

4.4 Processo de desenvolvimento

Segundo Sommerville (2011, p. 18), o processo de desenvolvimento de um software estar diretamente ligado a um conjunto de atividades que levam à produção de um produto. Para tal, existem muitos processos diferentes, mas todos devem incluir pelo menos quatro atividades que são fundamentais para a engenharia de software: Especificação, Implementação, Validação e Evolução.

Este trabalho procurou incluir no processo de desenvolvimento as quatro atividades, que foi: o levantamento das funcionalidades (*Product Backlog*), implementação das funcionalidades levantadas, validação através de testes de unidade e integração, e por fim, evolução do software com mudanças e inclusão de algumas funcionalidades.

O desenvolvimento do aplicativo Android foi realizado seguindo a metodologia de desenvolvimento ágil Scrum (Seção 2.5), com adaptações para atender as necessidades específicas do contexto no qual esta atividade foi executada. Além disso, a plataforma Trello¹⁴

¹³ <https://github.com/PhilJay/MPAndroidChart>

¹⁴ <https://trello.com>

foi escolhida como ferramenta principal para gerenciamento do processo de desenvolvimento. Esta escolha se deu pela grande flexibilidade fornecida pela ferramenta, possibilitando sua fácil adaptação a diferentes cenários de desenvolvimento, a sua simplicidade, ao caráter colaborativo da ferramenta e à possibilidade de uso da versão gratuita, que atende todos os requisitos necessários ao desenvolvimento deste trabalho.

5 RESULTADOS

Como descrito na Seção 4.4, o desenvolvimento da aplicação Android foi baseado na metodologia ágil Scrum, e foi subdividido em 4 etapas principais: (i) Especificação, (ii) Implementação, (iii) Validação e (iv) Evolução. As etapas (i), (ii) e (iii) serão apresentadas de forma detalhada nesta seção, incluindo os resultados obtidos em cada uma destas. A etapa (iv) será discutida na Seção 6.

5.1 Especificação

Nesta etapa, o *Scrum Master* realizou todo o levantamento de requisitos para especificação e posterior validação do aplicativo Android desenvolvido. Os requisitos coletados foram transformados no *Product Backlog* apresentado no APÊNDICE A - Product Backlog. Neste são listadas o conjunto de funcionalidades (*User Stories*) esperadas para o aplicativo Android, seguindo a nomenclatura USXX - TÍTULO, como exemplificado na Figura 13.

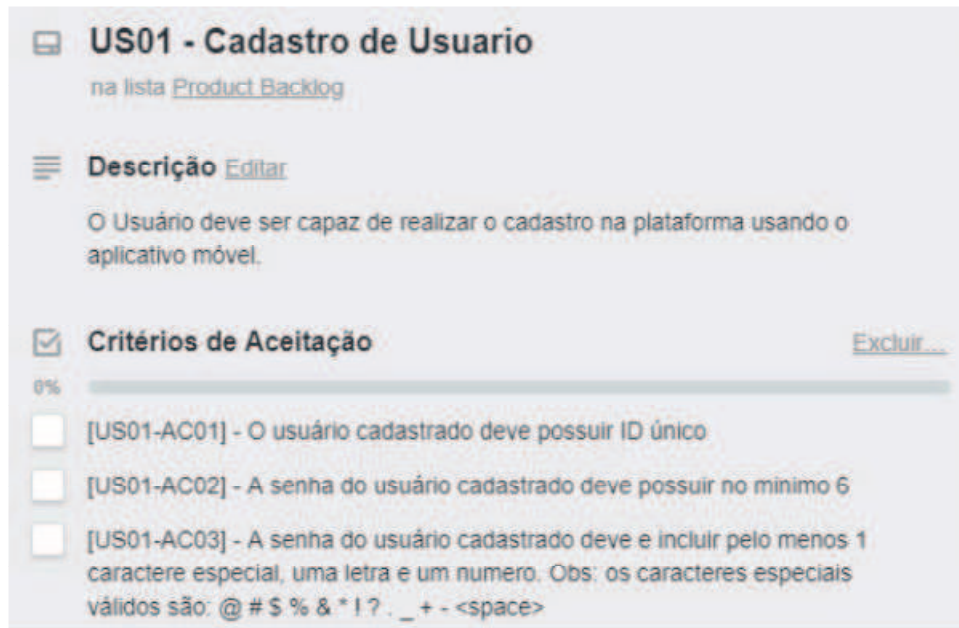
Figura 13 - Listagem de algumas *User Stories* do *Product Backlog*



Fonte: Próprio autor

Cada *User Story* possui uma breve descrição da funcionalidade esperada, assim como um conjunto de critérios de aceitação para validação da funcionalidade pela equipe de teste, após finalizada pela equipe de desenvolvimento, como pode ser observado na Figura 14.

Figura 14 – Detalhamento da *User Story* de cadastro de usuário



Fonte: Próprio autor

O conjunto de *User Stories* que compõe o *Product Backlog* do aplicativo Android são descritas a seguir:

- **US01 – Cadastro de Usuário.** O Usuário deve ser capaz de realizar o cadastro na plataforma usando o aplicativo móvel;
- **US02 – Autenticação de Usuário.** O acesso a plataforma HANIoT deve exigir a autenticação de usuários previamente cadastrados;
- **US03 – Cadastro de Dispositivos Pessoais de Saúde (PHDs).** O usuário cadastrado na plataforma HANIoT deve ser capaz de associar novos PHDs a sua conta;
- **US04 - Aquisição de Dados dos Dispositivos Pessoais de Saúde.** Os usuários autenticados no aplicativo móvel devem ser capazes de conectar dispositivos cadastrados na sua conta e receber medições realizadas por estes;
- **US05 - Armazenamento de Dados de Saúde.** Os dados de saúde dos usuários da plataforma HANIoT devem ser armazenados temporariamente pelo aplicativo móvel, e permanentemente pelo servidor Web;
- **US06 - Sincronização de Dados de Saúde do Usuário.** O aplicativo móvel deve sincronizar com o servidor os dados salvos localmente na conta do usuário autenticado no aplicativo móvel. A sincronização deve acontecer de forma automática e/ou sob demanda do usuário;

- **US07 - Visualização de Histórico de Dados de Saúde do Usuário.** O aplicativo móvel deve permitir a visualização do histórico dos dados de saúde associados ao usuário. A visualização inclui dados salvos localmente na conta do usuário autenticado no aplicativo móvel, na ausência de acesso à internet, e dados salvos no servidor Web.

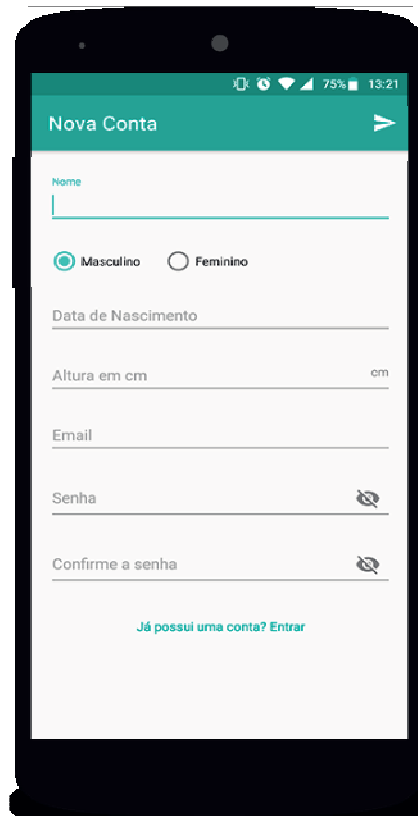
5.2 Implementação

Nesta seção, será detalhada a implementação de cada *User Story* definida no *Product Backlog*. A implementação da aplicação Android foi realizada no ambiente de desenvolvimento integrado do Android Studio.

5.2.1 US01 - Cadastro de usuário

Nessa *User Story* foi implementado o modelo de dados do usuário de acordo com os atributos definidos pela interface da API Rest (componente servidor da plataforma HANIoT, que não é incluído neste trabalho). Sendo assim, o usuário tem os seguintes atributos: id, nome, e-mail, senha, gênero, data de nascimento, altura e um *token* de acesso. Com base nos atributos definidos, foi desenvolvido a tela de cadastro do usuário conforme apresentado na Figura 15.

Figura 15 - Tela para cadastro de usuário

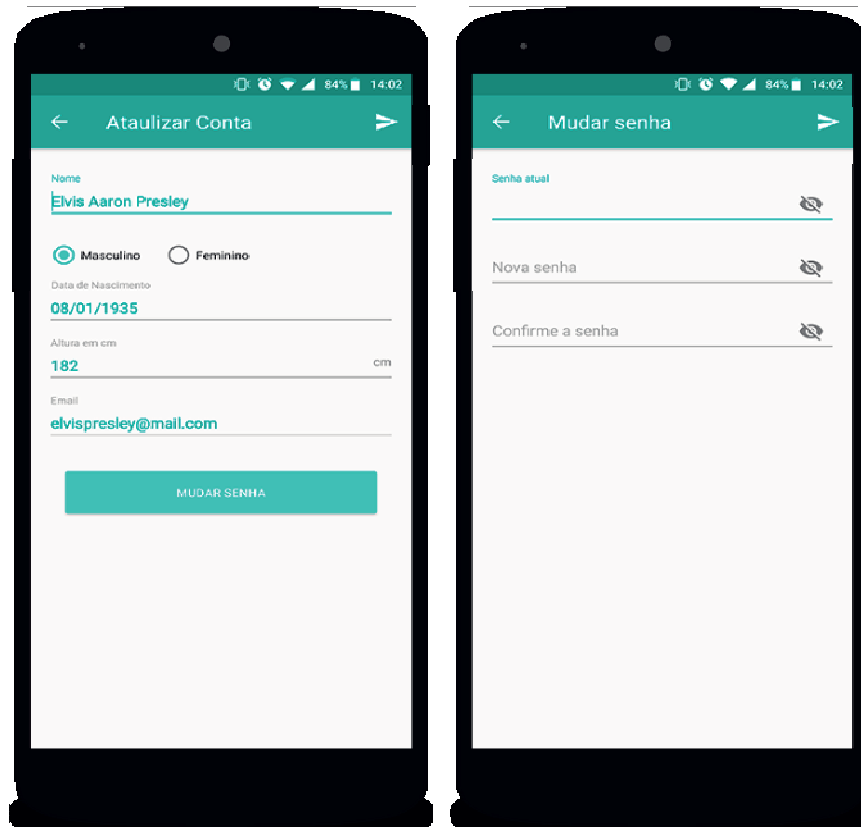


Fonte: Próprio autor

Um dos critérios de aceitação que foi definido, é que a senha do usuário cadastrado deve possuir no mínimo 6 caracteres e incluir pelo menos 1 caractere especial, uma letra e um número, sendo os caracteres especiais válidos: @ # \$ % & * ! ? . _ + - <espaço em branco>. Sendo assim, para validação da senha foi implementado uma função que utiliza a seguinte expressão regular `/((?=.*[a-zA-Z0-9])(?=.*[@#$%*!?._+-]).{6,})/g`, retornando verdadeiro se a senha satisfazer as condições e falso caso contrário. Após essa verificação, e outras como dados obrigatórios, o usuário é cadastrado com sucesso na plataforma.

Outro critério de aceitação definido foi a possibilidade de o próprio usuário, após cadastrado, poder realizar alterações das suas informações, assim como, a mudança de sua senha. Na Figura 16 podemos ver as telas implementadas para estas funcionalidades.

Figura 16 - Telas para alteração de dados cadastrais do usuário



Fonte: Próprio autor

5.2.2 US02 - Autenticação de usuário

O usuário após realização do cadastro é redirecionado para tela de autenticação, apresentada na Figura 17, onde poderá inserir e-mail e senha. Uma vez autenticado no sistema, esta tela não será exibida quando o aplicativo for aberto novamente, isso até que o próprio usuário opte por sair (por meio das configurações), ou o próprio aplicativo encerre a sessão, que de acordo com um dos critérios de aceitação deverá ter duração máxima de 1 dia. Entretanto, a funcionalidade de expiração da sessão não foi implementada nesta versão da aplicação, fazendo parte das tarefas de evolução.

Figura 17 - Tela para autenticação de usuário



Fonte: Próprio autor

5.2.3 US03 - Cadastro de dispositivos pessoais de saúde

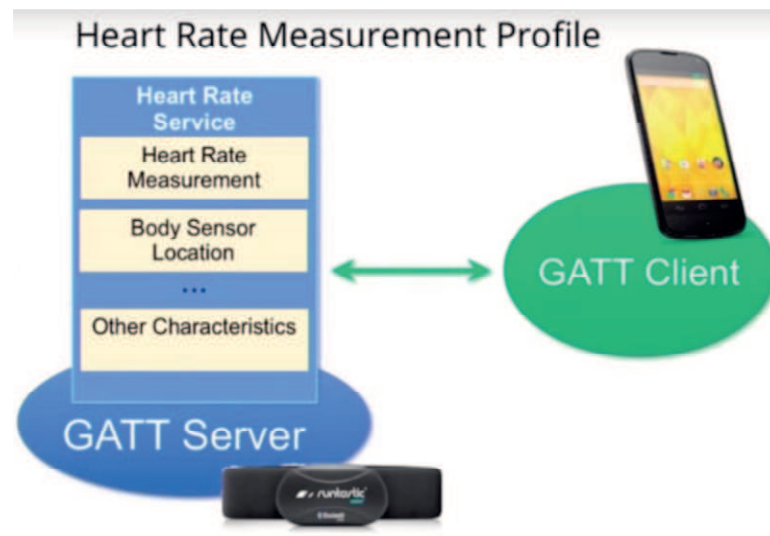
Nessa *User Story* era previsto a implementação da funcionalidade para cadastro dos dispositivos pessoais de saúde que foram selecionados e listados na Seção 4.2. No entanto, a funcionalidade foi colocada como sendo de baixa prioridade para esta versão do aplicativo Android, uma vez que todo o processo de captura de dados dos dispositivos pode ser realizado apenas com endereço físico *Bluetooth* dos mesmos. Sendo assim, optou-se por inserir manualmente as informações necessárias do dispositivo no código do aplicativo.

Vale ressaltar que apesar de terem sido selecionados apenas cinco dispositivos para validação da aplicação (*Philips Ear Thermometer, Accu-Chek Performa Connect, YUNMAI Mini 1501, Polar H10 Heart Rate Sensor e Xiaomi Mi Band 2*), é possível realizar a integração de qualquer outro dispositivo que siga a especificação oficial do Bluetooth SIG, independentemente do fabricante.

5.2.4 US04 - Aquisição de dados dos dispositivos pessoais de saúde

A aquisição de dados dos dispositivos pessoais de saúde é realizada através do protocolo de comunicação BLE. Para essa finalidade, foi implementado um módulo responsável pela comunicação entre os periféricos (dispositivos pessoais de saúde) e o dispositivo (celular e tablet) BLE.

Figura 18 - Comunicação entre cliente e servidor GATT



Fonte: (DEVELOPERS, 2013)

A Figura 18 ilustra o processo de comunicação entre um celular (cliente GATT) e um periférico de *Heart Rate* (servidor GATT). Após o cliente se conectar com o periférico através do endereço físico de seu *Bluetooth*, é possível realizar a troca de informação como por exemplo, o celular pode se inscrever na característica de UUID igual a 0x2A37 responsável pela medição da frequência cardíaca, e ser notificado a cada nova medição. Isso é possível porque de acordo com a especificação do Bluetooth SIG, essa característica possui propriedade de notificação.

O módulo implementado para essa *User Story* é um serviço que utiliza o próprio SDK do Android, o qual fornece as classes necessárias para implementação de soluções com o protocolo BLE. Esse serviço possui uma instância única e é inicializado quando a tela do dispositivo é aberta e encerrado quando a tela é fechada. Dentre os vários métodos que o módulo implementa, pode-se destacar:

- ***connect(final String address)*** - realiza a conexão entre o periférico e dispositivo através do endereço do periférico recebido como parâmetro;

- ***disconnect()*** - encerra a conexão aberta entre o periférico e dispositivo. Esse método é chamado quando o usuário fecha a tela que serve para capturar as medições do periférico, ou quando por algum motivo o Android termina o processo, algumas vezes por falta de recursos de memória;
- ***readCharacteristic(BluetoothGattCharacteristic)*** - solicita ao periférico os dados da característica passada como parâmetro, que deve ter propriedade de leitura. Por exemplo, podemos solicitar ao serviço de *Heart Rate* a leitura dos dados na característica de UUID 0x2A38. De acordo com o fabricante, esta característica informa a localização que o sensor deve ficar no corpo;
- ***writeCharacteristic(BluetoothGattCharacteristic)*** - envia informações para a característica do periférico passada como parâmetro. A informação é enviada em um *array* de *bytes* no atributo *byte[] mValue* da classe *BluetoothGattCharacteristic*;
- ***setCharacteristicNotification(BluetoothGattCharacteristic, byte[] descriptorValue, boolean enabled)*** - solicita ao periférico os dados da característica passada como parâmetro. O parâmetro *descriptorValue* indica se o processo será de notificação ou indicação. O último parâmetro, se *true*, ativa a notificação/indicação, se *false*, desativa.

Quando um dado é recebido do periférico ele vem em um objeto do tipo *BluetoothGattCharacteristic* que possui o atributo *mValue*, o qual é um *array* de *bytes*. Sendo assim, foi implementado métodos específicos para cada tipo de dispositivo para converter o *array* de *bytes* no formato específico de cada dispositivo em um JSON seguindo o modelo de dados da plataforma HANIoT. No APÊNDICE B - `Parser Heart Rate` é encontrado o código utilizado para realização do parser das medições dos dispositivos de frequência cardíaca.

Após esse processo de conversão dos dados, é realizada a transmissão (broadcast) do JSON por meio da interface *Context.sendBroadcast(Intent)* do Android. Logo, a tela que inicializou o serviço, ou qualquer outra classe que tenha se inscrito no serviço para receber a informação, recebe o dado já no formato padrão da plataforma HANIoT. Na Figura 19 podemos ver a tela do termômetro exibindo uma medição recebida do periférico.

Figura 19 - Tela do termômetro exibindo medição recebida



Fonte: Próprio autor

5.2.5 US05 - Armazenamento de dados de saúde

Para armazenamento de dados foi implementado um módulo específico que utiliza a biblioteca ObjectBox (Seção 4.3.1). Neste módulo são definidos todos os modelos de dados que o aplicativo implementa, por exemplo: Usuário, Dispositivo e Medição.

Cada modelo de dados possui uma classe responsável pelas operações de leitura, inserção, atualização e remoção das informações do modelo correspondente no banco de dados da aplicação Android. Essas classes são chamadas de *Data Access Object* (DAO).

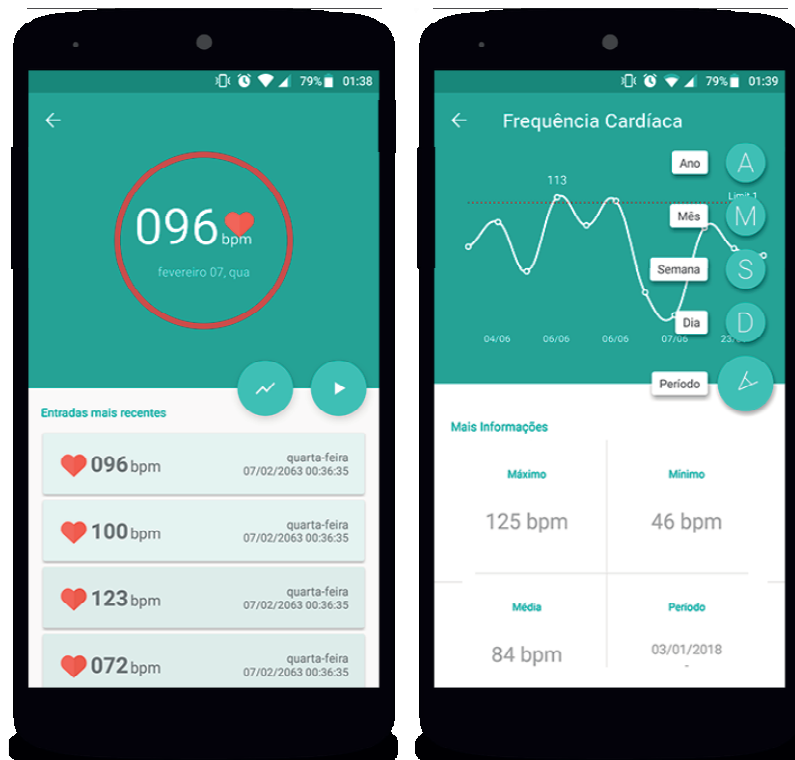
Em conformidade com um dos critérios de aceitação da *User Story*, as medições são salvas temporariamente no banco de dados do aplicativo Android, sendo removidas após a sincronização com o *backend* da plataforma HANIoT.

5.2.6 US06 - Visualização de histórico de dados de saúde

Para visualização de histórico de dados de saúde do usuário foram implementados os seguintes módulos:

- **Aquisição do Histórico de Dados de Saúde** - responsável por fazer a requisição dos dados desejados ao servidor, o qual responde com um JSON contendo o histórico de dados seguindo o modelo definido para a plataforma HANIoT. Essas informações são tratadas e convertidas para objetos, afim de serem facilmente utilizados no aplicativo Android;
- **Plotagem de Gráficos** – utiliza a biblioteca MPAndroidChart (Seção 4.3.4) para exibição do histórico de dados de saúde por meio de gráficos do tipo linha e barra, como pode ser observado na Figura 20 e na Figura 21.

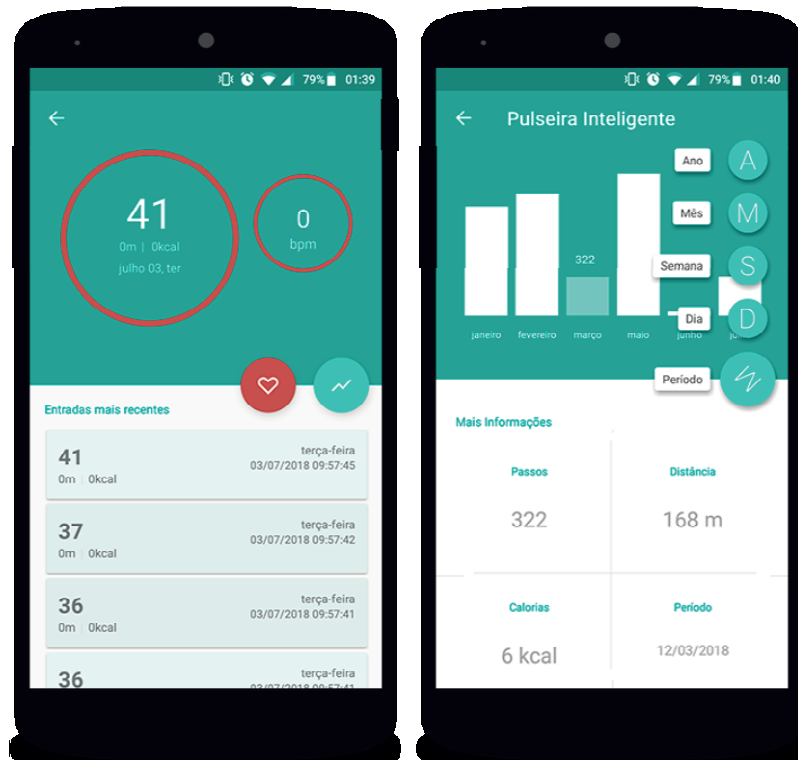
Figura 20 - Telas para visualização do histórico de frequência cardíaca



Fonte: Próprio autor

Dependendo do tipo de dado, a tela que exibe os gráficos das medições possibilita a filtragem do histórica exibido de acordo com dia, semana, mês ou ano (Figura 21).

Figura 21 - Telas para visualização do histórico de smartband



Fonte: Próprio autor

5.2.7 US07 - Sincronização de dados de saúde

Para essa *User Story* foram implementados dois módulos, um para tratamento de requisições HTTP e outro para sincronização das medições com o servidor.

O módulo para tratamento de requisições HTTP foi implementado utilizando a biblioteca OkHttp (Seção 4.3.2). Este módulo encapsula toda lógica necessária para realizar as requisições à API Rest fornecida pelo *backend* da plataforma HANIoT. Além disso, o módulo implementa a camada de segurança utilizando o protocolo SSL, que estabelece um canal criptografado para troca de informações entre o aplicativo e servidor.

Já o módulo de sincronização, é o responsável por recuperar todas as medições armazenadas temporariamente no banco de dados do aplicativo Android, converte-las para um JSON seguindo o modelo definido para a plataforma HANIoT e envia-las para a API Rest do *backend* HANIoT usando o módulo de tratamento de requisições HTTP. A conversão das medições para JSON foi realizada com auxílio da biblioteca Gson (Seção 4.3.3) e da classe *JSONObject* do SDK Android.

Quando a sincronização é realizada com sucesso, o *backend* retornará uma resposta que contém o código de *status* igual a 201 e o módulo de sincronização remove todas as medições

enviadas do banco de dados do aplicativo Android, satisfazendo o critério de aceitação estabelecido para esta *User Story*. Caso contrário, se o servidor retornar um código de *status* diferente de 201, significa que a requisição não foi realizada com sucesso, seja porque aconteceu algum erro no servidor, ou por falta de conexão com a internet. Sendo assim, as medições são mantidas no banco de dados do aplicativo Android, até que sejam enviadas em tentativas posteriores.

Existem quatro situações em que o módulo de sincronização de dados é chamado, são elas:

- I. Quando a tela de *Splash Screen* (Tela de Abertura) é inicializada. O tempo de exibição desta tela depende do tempo necessário para a sincronização dos dados com o servidor. Sendo assim, quando não existe dados a serem enviados esta tela não é apresentada ao usuário;
- II. Quando o usuário abre a tela de captura de dados do dispositivo;
- III. Quando o usuário está na tela de captura de dados do dispositivo, e esta recebe uma nova medição;
- IV. Quando o usuário clica no botão de sincronização que fica na barra superior da tela principal do aplicativo (Figura 22).

Figura 22 - Tela principal do aplicativo HANIoT



Fonte: Próprio autor

5.3 Validação

O processo de validação se deu basicamente com a realização de testes de unidade utilizando a biblioteca Robolectric¹⁵ e com a realização de testes de integração utilizando a biblioteca espresso¹⁶. A biblioteca Robolectric possibilita a execução dos testes dentro da JVM, tornando o processo mais rápido, uma vez que a emulação ou compilação do aplicativo em um dispositivo real não é necessária. A biblioteca espresso é fornecida pela própria Google, e simula a interação do usuário com o aplicativo, que apesar de ser teste mais custoso é de grande importância.

Os erros encontrados pela equipe de teste foram registrados no Trello em uma lista exclusiva para “Problemas de Validação”, como apresentado na Figura 23. Pelo curto tempo de execução do trabalho, não foram realizadas *Sprints* para correção dos problemas relatados, que devem ser endereçados nas próximas versões do aplicativo.

Figura 23 - Listagem problemas de validação



Fonte: Próprio autor

¹⁵ <http://robolectric.org>

¹⁶ <https://developer.android.com/training/testing/espresso>

6 CONCLUSÃO

Devido ao alto crescimento da IoT na área de saúde, novos dispositivos constantemente são lançados no mercado por fabricantes diversos. A solução proposta neste trabalho atingiu o objetivo de integração de dispositivos pessoais de saúde de diferentes fabricantes à plataforma HANIoT por meio do desenvolvimento de um aplicativo Android para coleta e gerenciamento de dados de saúde de dispositivos que utilizam a tecnologia *Bluetooth Low Energy*. O aplicativo é um dos componentes da plataforma HANIoT, também composta pelos componentes Servidor e Dashboard, os quais não foram desenvolvidos no escopo deste trabalho.

O aplicativo foi desenvolvido procurando seguir boas práticas de desenvolvimento de *software*, com alto nível de modularização, de modo a tornar a solução reutilizável e facilitar a inserção de novas funcionalidades. O módulo para requisições HTTP/HTTPS é o responsável por toda a comunicação com o componente servidor da plataforma HANIoT. O módulo de persistência é o responsável em salvar os dados de usuário, dispositivos e medições no banco de dados do celular/tablet. O módulo de sincronização é o responsável por fazer o tratamento das medições salvas no banco de dados do celular e enviar para o servidor no formato padrão da plataforma HANIoT. Por fim, os módulos para visualização de histórico de dados de saúde são responsáveis por recuperar os dados salvos no servidor e realizar a exibição dos gráficos.

Utilizando os dados coletados de diferentes dispositivos pessoais de saúde e integrados à plataforma HANIoT pelo aplicativo Android desenvolvido neste trabalho, pode-se realizar estudos e implementar funcionalidades relacionadas à análise de dados e monitoramento de usuários por profissionais de saúde, auxiliando na mudança de hábitos e prevenção de problemas de saúde.

Durante a pesquisa e desenvolvimento foi observada a possibilidade de coletar dados de alguns dispositivos que inicialmente não foram inseridos no escopo deste trabalho. Apesar de não seguirem a especificação do Bluetooth SIG, os fabricantes destes dispositivos disponibilizam os dados coletados por meio de uma API Rest aberta. Como exemplo, pode-se destacar os dados coletados por meio de dispositivos da FitBit. A integração com a plataforma FitBit será uma funcionalidade adicionada em versões futuras do aplicativo Android.

Desta forma, com a realização de trabalhos futuros espera-se melhorias nas funcionalidades já exploradas, como o suporte a outros tipos de medições e dispositivos; a correção de problemas de validação encontrados na versão atual; a implementação de funcionalidades que foram retiradas do escopo deste trabalho, como o cadastro de dispositivos; a integração do aplicativo com funcionalidades que serão implementadas na plataforma

HANIoT, como análise de dados e inteligência artificial; e a validação da solução em cenários reais por meio da realização de estudos piloto em parceria com profissionais de saúde.

REFERÊNCIAS

- BHARGAVA, M. **IoT Projects with Bluetooth Low Energy**: Harness the power of connected things. 1. ed. United States of America: Packt Publishing, 2017.
- BUYYA, R.; DASTJERDI, A. V. **Internet of Things**: Principles and Paradigms. 1. ed. United States of America: Elsevier Inc, 2016.
- CRUZ, R. R. D. **Como a saúde pode se beneficiar com a Internet das Coisas**, 2015. Disponível em: <<https://saudebusiness.com/noticias/como-saude-pode-se-beneficiar-com-internet-das-coisas/>>. Acesso em: 30 jul. 2018.
- DEVELOPERS, A. **DevBytes**: Bluetooth Low Energy API in Android 4.3, 2013. Disponível em: <<https://youtu.be/vUcFB1Qypg8>>. Acesso em: 05 ago. 2018.
- DEVELOPERS, A. **Versões da plataforma**, 2018. Disponível em: <<https://developer.android.com/about/dashboards/>>. Acesso em: 05 ago. 2018.
- DIAS, R. R. D. F. **Internet das Coisas sem mistérios**: uma nova inteligência para os negócios. 1. ed. São Paulo: Netpress Books, 2016.
- EVANS, D. **A Internet das Coisas**: Como a próxima evolução da Internet está mudando tudo. [S.l.]: Cisco IBSG, 2011.
- FILHO, M. F. **Internet das coisas**: livro digital. 1. ed. Palhoça: UnisulVirtual, 2016.
- GARGENTA, M. **Learning Android**. 1. ed. United States of America: O'Reilly, 2011.
- GLAUBER, N. **Dominando o Android**: Do básico ao avançado. 1. ed. São Paulo: Novatec Editora Ltda, 2015.
- GOOGLE. **Architecture Android Open Source Project**, 2017. Disponível em: <<https://source.android.com/devices/architecture>>. Acesso em: 15 fev. 2018.
- HEYDON, R. **Bluetooth Low Energy**: The Developer's Handbook. 1. ed. United States of America: Prentice Hall, 2012.
- ITU-T. **Recommendation ITU-T Y.2060: Overview of the Internet of things**. Telecommunication Standardization Sector. [S.l.], 2012.
- LATAM, M. M. **Saúde Digital**. Disponível em: <<http://forumsaudedigital.com.br/internet-das-coisas-e-o-avanco-da-tecnologia-na-area-de-saude/>>. Acesso em: 29 jul. 2018.

LECHETA, R. R. **Android**: Aprenda a criar aplicações para dispositivos móveis com Android SDK. 4. ed. São Paulo: Novatec Editora Ltda, 2009.

NETO, E. S. D. S. **Desenvolvimento de um componente servidor para gerenciamento de dados médicos**. UEPB. Campina Grande. 2018.

OBJECTBOX. **ObjectBox**, 2018. Disponível em: <<https://objectbox.io/>>. Acesso em: 06 ago. 2018.

PESSOA, L. **Visão Técnica do Bluetooth Smart**, 2016. Disponível em: <<https://www.embarcados.com.br/bluetooth-smart-visao-tecnica/>>. Acesso em: 30 jul. 2018.

PIRES, J. **O que é API? REST e RESTful? Conheça as definições e diferenças!**, 2017. Disponível em: <<https://becode.com.br/o-que-e-api-rest-e-restful/>>. Acesso em: 06 ago. 2018.

ROUSE, M. IoT Agenda. **IoMT (Internet of Medical Things) or healthcare IoT**, 2015. Disponível em: <<https://internetofthingsagenda.techtarget.com/definition/IoMT-Internet-of-Medical-Things>>. Acesso em: 07 ago. 2018.

SAUDATE, A. **REST**: Construa API's inteligentes de maneira simples. São Paulo: Caso do Código, 2014.

SCHWABER, K.; SUTHERLAND, J. **Um guia definitivo para o Scrum**: As regras do jogo. [S.l.]: [s.n.], 2013.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

SQUARE. **OkHttp**, 2018. Disponível em: <<http://square.github.io/okhttp/>>. Acesso em: 06 ago. 2018.

STATCOUNTER GLOBALSTATS. **Operating System Market Share Worldwide**. Disponível em: <<http://gs.statcounter.com/os-market-share>>. Acesso em: 15 fev. 2018.

SUTHERLAND, J. **SCRUM**: A arte de fazer o dobro do trabalho na metade do tempo. 1. ed. São Paulo: LeYa, 2014.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 3. ed. São Paulo: Pearson Prentice Hall, 2009.

TOWNSEND, K. **Introduction to Bluetooth Low Energy**, 30 jul. 2015. Disponível em: <<https://learn.adafruit.com/introduction-to-bluetooth-low-energy>>.

TOWNSEND, K. et al. **Getting started with bluetooth low energy**. United States of America: O'Reilly, 2014.

APÊNDICE A - Product Backlog

Título	Descrição	Critérios de Aceitação
<p>US01 - Cadastro de Usuário.</p>	<p>O Usuário deve ser capaz de realizar o cadastro na plataforma usando o aplicativo móvel.</p>	<p>[US01-AC01] - O usuário cadastrado deve possuir ID único.</p> <p>[US01-AC02] - A senha do usuário cadastrado deve possuir no mínimo 6 caracteres.</p> <p>[US01-AC03] - A senha do usuário cadastrado deve incluir pelo menos 1 caractere especial, uma letra e um número. Sendo os caracteres especiais válidos: @ # \$ % & * ! ? . _ + - <space>.</p> <p>[US01-AC04] - O aplicativo móvel deve fornecer a interface para cadastro de usuários.</p> <p>[US01-AC05] - As informações do cadastro do usuário devem ser criptografadas para transmissão e salvamento no servidor.</p> <p>[US01-AC06] - O servidor deve realizar a validação do cadastro do usuário.</p> <p>[US01-AC07] - A senha do usuário não pode conter acentuação.</p> <p>[US01-AC08] - O aplicativo deve fornecer interface para atualização/modificação de dados cadastrais do usuário.</p> <p>[US01-AC09] - O usuário, e apenas ele, deve ser capaz de realizar a exclusão da sua conta (Login), possibilitando ainda que posteriormente o mesmo e-mail torne a ser cadastrado.</p> <p>[US01-AC10] - O usuário já cadastrado, e apenas ele, deve ser capaz de realizar alterações das informações fornecidas durante o cadastro, exceto o password e e-mail.</p> <p>[US01-AC11] - O usuário já cadastrado, e apenas ele, deve ser capaz de realizar alterações do password.</p> <p>[US01-AC12] - O usuário deve ser capaz de recuperar a senha de login, em caso de esquecimento a partir do e-mail.</p>

<p>US02 - Autenticação de Usuário.</p>	<p>O acesso a plataforma HANIoT deve exigir a autenticação de usuários previamente cadastrados.</p>	<p>[US02-AC01] - O aplicativo móvel deve fornecer interface para autenticação do usuário normal.</p> <p>[US02-AC02] - A autenticação dos usuários deve possuir validade máxima de 1 dia.</p> <p>[US02-AC03] - O servidor deve validar a autenticação com base nas informações armazenadas para usuários cadastrados.</p> <p>[US02-AC04] - As informações de autenticação devem ser transmitidas por meio de conexão criptografada entre aplicativo móvel e servidor.</p>
<p>US03 - Cadastro de Dispositivos Pessoais de Saúde (PHD).</p>	<p>O usuário cadastrado na plataforma HANIoT deve ser capaz de associar novos PHDs a sua conta.</p>	<p>[US03-AC01] - O usuário comum deve ser capaz de cadastrar dispositivos suportados na plataforma.</p> <p>[US03-AC02] - O servidor deve realizar a validação do cadastro de dispositivos.</p> <p>[US03-AC03] - O usuário comum deve ser capaz de remover dispositivos.</p> <p>[US03-AC04] - O usuário comum deve ser capaz de realizar o pareamento quando requerido pelo dispositivo dentro da plataforma.</p> <p>[US03-AC05] - O dispositivo cadastrado deve ser associado ao usuário comum.</p> <p>[US03-AC06] - Um dispositivo pode estar associado a um ou mais usuários.</p> <p>[US03-AC07] - O servidor só deve remover um dispositivo da base dados se estiver associado com apenas um usuário, caso contrário remover apenas a associação.</p>
<p>US04 - Aquisição de dados dos PHDs.</p>	<p>Os usuários autenticados no aplicativo móvel devem ser capazes de conectar dispositivos cadastrados na sua conta e receber medições realizadas por estes.</p>	<p>[US04-AC01] - O Usuário logado no aplicativo móvel deve ser capaz de se conectar a dispositivos BLE cadastrados na sua conta.</p> <p>[US04-AC02] - O Usuário logado no aplicativo móvel deve ser capaz de receber medições de dispositivos BLE conectados.</p>

<p>US05 Armazenamento de dados de saúde.</p>	<p>Os dados de saúde dos usuários da plataforma HANIoT devem ser armazenados temporariamente pelo aplicativo móvel, e permanentemente pelo servidor WEB.</p>	<p>[US05-AC01] - O servidor deve realizar o armazenamento permanente de dados do usuário em um banco de dados.</p> <p>[US05-AC02] - O aplicativo móvel deve realizar o armazenamento temporário de dados do usuário em um banco de dados.</p> <p>[US05-AC03] - O modelo de dados do servidor e do aplicativo móvel deve ser validado de acordo com as consultas e operações previstas.</p>
<p>US06 - Sincronização de dados de saúde do usuário.</p>	<p>O aplicativo móvel deve sincronizar com o servidor os dados salvos localmente na conta do usuário logado. A sincronização deve acontecer de forma automática e/ou sob demanda do usuário.</p>	<p>[US06-AC01] - Caso tenha conexão com o servidor, o aplicativo móvel deve sincronizar as medições do usuário logado sempre que a tela de dispositivos cadastrados for aberta.</p> <p>[US06-AC02] - Caso tenha conexão com o servidor, o aplicativo móvel deve sincronizar as medições do usuário logado sempre que o usuário cadastrado solicitar.</p> <p>[US06-AC03] - Caso tenha conexão com o servidor, o aplicativo móvel deve sincronizar imediatamente as medições realizadas pelo usuário logado por meio de dispositivos cadastrados.</p> <p>[US06-AC04] - A sincronização de medições com servidor deve ser feita por meio de conexão criptografada.</p> <p>[US06-AC05] - As medições enviadas pelo aplicativo móvel devem ser validadas pelo servidor quanto ao usuário, tipos de medição, e demais informações obrigatórias.</p> <p>[US06-AC06] - Após sincronização com o servidor as medições armazenadas localmente devem ser excluídas.</p>

<p>US07 - Visualização de histórico de dados de saúde do usuário.</p>	<p>O aplicativo móvel deve permitir a visualização do histórico dos dados de saúde associados ao usuário. A visualização inclui dados salvos localmente na conta do usuário logado no aplicativo móvel, na ausência de acesso a internet, e dados salvos no servidor WEB.</p>	<p>[US07-AC01] - O Usuário logado no aplicativo móvel deve ser capaz de visualizar o gráfico do seu histórico para um determinado tipo de medição (temperatura corporal, peso, batimento cardíaco, etc.) e intervalo de tempo (dia, mês e ano).</p> <p>[US07-AC02] - O Usuário logado no aplicativo móvel deve ser capaz de visualizar os valores das últimas medições realizadas por um dispositivo cadastrado.</p> <p>[US07-AC03] - O servidor deve realizar a validação dos usuários para os diferentes tipos de consulta.</p> <p>[US07-AC04] - A consulta ao histórico de medições salvos no servidor deve ser feita por meio de conexão criptografada entre aplicativo móvel e servidor.</p>
---	---	---

APÊNDICE B – Parser Heart Rate

```

package br.edu.uepb.nutes.haniot.parse;

import android.bluetooth.BluetoothGattCharacteristic;

import org.json.JSONException;
import org.json.JSONObject;

import br.edu.uepb.nutes.haniot.utils.DateUtils;

/**
 * Parse for heart rate.
 *
 * @author Douglas Rafael <douglas.rafael@nutes.uepb.edu.br>
 * @version 1.0
 * @copyright Copyright (c) 2017, NUTES UEPB
 */
public class GattHRParser {
    private static final byte HEART_RATE_VALUE_FORMAT = 0x01; // 1 bit

    /**
     * Parse for the POLAR device, according to GATT.
     * Supported Models: H7, H10.
     *
     * {@link <https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.characteristic.heart_rate_measurement.xml>}
     *
     * @param c
     * @return JSONObject
     * @throws JSONException
     */
    public static JSONObject parse(
        final BluetoothGattCharacteristic c) throws JSONException {

        JSONObject result = new JSONObject();
        int offset = 0, heartRateValue = 0;

        final int flags = c.getIntValue(
            BluetoothGattCharacteristic.FORMAT_UINT8,
            offset++
        );

        /*
         * false - Heart Rate Value Format is set to UINT8.
         * true - Heart Rate Value Format is set to UINT16.
         */
        final boolean value16bit = (flags & HEART_RATE_VALUE_FORMAT) > 0;

        // heart rate value is 8 or 16 bit long
        heartRateValue = c.getIntValue(
            value16bit ? BluetoothGattCharacteristic.FORMAT_UINT16 :
            BluetoothGattCharacteristic.FORMAT_UINT8, offset++
        );

        /**
         * Populating the JSON
         */
        result.put("heartRate", heartRateValue);
        result.put("heartRateUnit", "bpm");
        result.put("timestamp", DateUtils.getCurrentDatetime());

        return result;
    }
}

```