PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

MONITORAMENTO PERSONALIZADO PARA ALERTA DE EMERGÊNCIAS UTILIZANDO SMARTWATCH

GABRIEL DE PAIVA CASTRO

GOIÂNIA
2022

GABRIEL DE PAIVA CASTRO

MONITORAMENTO PERSONALIZADO PARA ALERTA DE EMERGÊNCIAS UTILIZANDO SMARTWATCH

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador:

Prof. M.E. Marcelo Antonio Adad de Araújo

GOIÂNIA

2022

GABRIEL DE PAIVA CASTRO

MONITORAMENTO PERSONALIZADO PARA ALERTA DE EMERGÊNCIAS UTILIZANDO SMARTWATCH

Trabalho de Conclusão de Curso aprovado em sua forma final pela Escola Politécnica, da Pontifícia Universidade Católica de Goiás, para obtenção do título de Bacharel em Engenharia de Computação, em _____/_____/_____.

_____

Prof. Ma. Ludmilla Reis Pinheiro dos Santos
Coordenadora de Trabalho de Conclusão de Curso

Banca examinadora:

_____

Orientador: Prof. Marcelo Antonio Adad de Araújo M.E.E.

_____

Prof. Carlos Alexandre Ferreira de Lima M.E.E.

_____

Prof. Mírian Sandra Rosa Gusmão M.E.A

GOIÂNIA
2022

DEDICATÓRIA

Dedico à minha família, em especial a meus pais que sempre foram uma inspiração de luta e esforço constantes.

A todos que me ajudaram e estiveram presentes em minha trajetória acadêmica.

AGRADECIMENTOS

Aos meus pais Cristina Vaz e João César pelo incentivo e determinação à vida acadêmica, com apoio incondicional.

Ao meu professor orientador Marcelo Antonio Adad de Araújo pela disponibilidade e paciência para me educar e instruir durante não só no período de síntese de TCC, mas também nos semestres antecedentes.

RESUMO

O presente trabalho, tem como objetivo auxiliar pessoas por meio de sensores já existentes nos dispositivos de smartband e smartwatch, com conexão *bluetooth* à um aplicativo Android mobile para acesso às informações e possíveis alarmes e localização para auxílio em casos de emergência. Para tal serão analisadas as mensurações colocando intervalos padrões do usuário, juntamente com um aplicativo mobile e uma base de dados NoSQL, para auxílio com GPS, alertas e ligações.

Palavras-Chave: *Android; Smartwatch; Bluetooth; sensores; cuidados; frequência cardíaca.*

ABSTRACT

The present academic work seeks to help people supported by the existing sensors in smartband and smartwatch devices, with a bluetooth connection to an Android mobile application to access information and possible generate alarms and location to help in emergencies.

For this, user's standard intervals and measurements will be record with a mobile application and a NoSQL database, to help with GPS, alarms and connection.

Keywords: *Android; Smartwatch; Bluetooth; sensors; care; heart rate.*

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

Me(a) Mestre(a)

M.E.E Mestre em engenharia elétrica

M.E.A Mestre em engenharia de automação

Prof. Professor

LISTA DE SIGLAS

AoT - *Analytics of Things*

API - *Application Programming Interface*

APP – *Application*

ATT – *Attribute Protocol*

BLE - *Bluetooth Low Energy*

BPM – Batimentos Por Minuto

FC - Frequência Cardíaca

GAP - *Generic Access Profile*

GATT - *Generic Attribute Profile*

GPS - *Global Position System*

HTTP - *Hypertext Transfer Protocol*

IDE - *Integrated Development Environment*

IoT - *Internet of Things*

IU - User Interface

JSON - *JavaScript Object Notation*

MAC - *Media Access Control*

NoSQL - *Not Only SQL*

PUC – Pontifícia Universidade Católica

SDK - *Software Development Kit*

SO – Sistema Operacional

SOBRAC - Sociedade Brasileira de Arritmias Cardíacas

TCC – Trabalho de Conclusão de Curso

UUID - *Universally Unique Identifier*

VR – *Virtual Reality*

Wi-fi - *Wireless Fidelity*

SUMÁRIO

1 INTRODUÇÃO

Com a evolução dos aparelhos vestíveis, foram implementados em sua arquitetura sensores (PÉREZ; RODRÍGUEZ; GAGO, 2016) com potencial de analisar a performance de seus usuários em suas atividades físicas, desde seus passos, até análises mais precisas para cada tipo de exercício. Juntamente com esses sensores de movimento, também foram integrados sensores no âmbito da saúde, capazes de verificar a porcentagem de oxigênio no sangue, a frequência cardíaca e a pressão arterial (PÉREZ; RODRÍGUEZ; GAGO, 2016), ampliando e dando maior acessibilidade de medir à saúde da população com um simples acessório computadorizado (SILVA; FERNANDES; LINS, 2020).

Ao analisar esses dados, é possível verificar padrões e intervir com cuidados médicos, nos momentos de urgência, em usuários com maior necessidade de cuidados, como idosos e pessoas com deficiência cardíaca (SILVA; FERNANDES; LINS, 2020). Esses sensores, integrados aos relógios aliados à Internet das Coisas e a um banco de dados reativo e com informações em tempo real, permitem que essa intervenção de urgência seja mais precisa e pode melhorar qualidade de vivência desses usuários e até ampliar sua expectativa (DAVENPORT, 2015).

Com base nessa possibilidade, o presente trabalho busca analisar a seguinte questão de pesquisa:

- **É possível, por meio da mensuração, de sensores de análise vital do smartwatch, juntamente com um aplicativo mobile, gerar alertas de emergências assertivas?**

O trabalho aqui descrito inicia-se com uma introdução no capítulo 1 com os objetivos gerais e específicos principais do mesmo juntamente com a justificativa para tal pesquisa e implementação. Seguida dos métodos para a síntese do projeto, os resultados esperados para projetos futuros. Finalizado com a lista de atividades e o cronograma para implementação destas.

O capítulo 2 é empenhado na revisão bibliográfica, ingressando nos conhecimentos necessários para o entendimento geral do trabalho.

O capítulo 3 aprofunda de maneira direta nos conceitos com maior importância para a obtenção dos resultados.

O capítulo 4 faz menção aos itens utilizados no projeto para criação do aplicativo, finalizando com o capítulo 5 onde são concluídas as ideias do projeto e citados os trabalhos futuros.

1.1 Objetivos

Esta seção descreve os objetivos gerais e específicos do trabalho.

1.1.1 Objetivo Geral

O presente trabalho tem por objetivo mensurar padrões de batimentos cardíacos, com o auxílio de sensor de *smartwatch/smartband*, com a visualização dos dados no aplicativo mobile, com utilização de alerta, para o cuidador da área da saúde cadastrado fazer monitoramento, em situações de emergência.

1.1.2 Objetivos Específicos

- Implementar análises de padrões em um *smartwatch/smartband*;
- Analisar os padrões das medidas definindo seus níveis mais altos e mais baixos da frequência cardíaca;
- Enviar dados por *bluetooth* para o dispositivo mobile;
- Gravar padrões de análise e verificar os sinais em médias regulares de tempo;
- Permitir monitoramento remoto de BPM,
- Gerar alerta em caso de frequência alterada;

1.2 Justificativa

O presente trabalho justifica-se na possibilidade de mediar urgências com maior probabilidade de sucesso e rapidez por meio de um dispositivo não invasivo e de fácil portabilidade, utilizando uma tecnologia já consagrada no mercado e de acesso relativamente fácil.

O trabalho gerará ao usuário uma possibilidade de medição relativamente rápida, com maior conforto, facilidade e liberdade aos seus usuários.

1.3 Métodos

Em primeira instância, foi desenvolvida uma revisão bibliográfica em livros, revistas, artigos científicos e meios de informações confiáveis. Após essa pesquisa, foi analisado o sistema no qual é desenvolvido o aplicativo, juntamente com a aquisição de dados realizados

pelo dispositivo *smartwatch/smartband.* Juntamente com isso, foi desenvolvido o aplicativo mobile para mostrar e identificar urgências.

1.4 Resultados esperados

É esperado que o projeto ajude pessoas que necessitam de cuidados constantes, para que consigam ser mais independentes e ter mais privacidade em suas vidas, ao mesmo tempo que consigam rápida ajuda e auxílio em caso de emergências.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo contém conceitos importantes das funcionalidades dos requisitos ímpares para a execução do projeto aqui descrito, introduzindo a internet das coisas, seguido pela introdução básica de *bluetooth low energy* e finalizando com os *smartwatches.* A internet das coisas, implementada juntamente com o *Bluetooth Low Energy,* utiliza os sensores do *smartwatch* permitindo gravar os dados no *Google Firebase* identificando os padrões de limites da frequência cardíaca, evidenciando ao usuário pelo aplicativo de *smartphone* em casos de urgência.

2.1 *Internet of things*

A computação chamada de ubíqua tem por definição como uma computação que sai dos âmbitos de estações fixas e se torna pervasiva na vida das pessoas (ARAÚJO 2003). Segundo Araújo, foram as evoluções de componentes tecnológicos como as de microprocessadores, de rádio e de dispositivos pessoais, que levaram à ubiquidade para que dispositivos inteligentes e móveis possam auxiliar as capacidades humanas de forma universal e transparente.

Com a evolução da *internet*, muito do que se vivia do cotidiano antigamente foi readaptado para as novas possibilidades de fontes e conexões de informação, facilitando cada dia mais as tarefas diárias que antigamente podiam durar dias. Nesse mesmo modelo de intercomunicações de pessoas, foi-se promovendo uma nova tecnologia adaptada e conectada as “coisas” do nosso dia-a-dia. (SANTOS, 2016).

A Internet das Coisas, ou *Internet of Things* (IoT), nada mais é do que a comunicação entre dispositivos que antes não eram comunicáveis. Segundo Magrani (2018) a IoT, nada mais é que um ecossistema de informações de objetos físicos, com computação onipresente (ubíqua), capaz de facilitar o cotidiano de um indivíduo, por meio de interações entre si, que antes gastariam mais tempo e esforço para serem feitas.

De acordo com a Oracle, esse tipo de tecnologia existe graças a alguns fatores, sendo eles: o acesso à tecnologia de sensores de baixo custo e baixa potência, permitindo que suas medições sejam contínuas e duradouras, já que não gastam tanta energia, e a conectividade, que foi possível graças a protocolos de rede da Internet que facilitam a conexão de sensores à nuvem.

A Oracle ainda afirma que a previsão de dispositivos conectados no ano de 2025 seja de 22 bilhões, um crescimento notável em comparação à de previsão de 7 bilhões em 2020.

Com essa conexão de dados entre “coisas”, todo tipo de informação pode ser obtido e analisado de forma inteligente e que pode abranger para novos ideais e possibilitar novas visões de mundo que não se tinha antes (SANTOS, 2016).

Para Davenport (2015), há ainda a soma dos conceitos de IoT com análise procedural (AoT - *Analytics of Things*). Essa análise de dados se ramifica em várias possibilidades, sendo elas: entender padrões e analisar o porquê de variações, detectar anomalias, recurso de manutenção preditivo, otimização, prescrição e consciência situacional. Esses tipos de análise se mantêm no âmbito de implementar um sensor ou medidor em uma área que já é funcional para analisar dados que antes não se tinha.

Essas análises significativas transformam o mundo com mudanças inteligentes e precisas a partir de perspectivas únicas e que só são possíveis graças a IoT.

2.2 *Bluetooth Low Energy*

Introduzido pela Ericsson na década de 1990, o *Bluetooth*, é uma das mais utilizadas formas de tecnologia de curto alcance, menor que do *Wi-fi*, mas também mais econômica em questão de energia utilizada (SANTOS 2016). Esse tipo de Bluetooth foi e continua sendo utilizado pelo mundo todo, porém a tecnologia de bluetooth para a internet das coisas, precisa ser ainda mais rentável, já que na IoT, a mensuração e utilização de certos tipos de sensores ou medidores precisa ser utilizada constantemente, o que pede uma maior de economia de energia das partes que à utilizam.

Bluetooth Low Energy (BLE) é uma melhor utilização da já recorrente tecnologia embutida em vários dispositivos do nosso dia a dia, com uma maior economia de energia. De acordo com a Bluetooth, o BLE transmite em menos canais que sua versão mais antiga, chamada por eles de *Bluetooth Classic*, porém expandindo sua topologia que até então era “ponto a ponto”, para *broadcasting*, o que permite mais flexibilidade e velocidade, já que se conecta a vários dispositivos diferentes simultaneamente, e mais recentemente na topologia *mesh,* o que permite descentralização de controle e comunicação entre dispositivos, como apresentado nas figuras 1 e 2.

Figura 1 - Comparativo *Bluetooth Classic* e *Bluetooth Low Energy*.

Fonte: Adaptado de *Bluetooth*

Figura 2 - Comparativo *Bluetooth Classic* e *Bluetooth Low Energy* especificado.

	Bluetooth de baixa energia (LE)	Bluetooth clássico
Faixa de frequência	Banda ISM de 2,4 GHz (2,402 – 2,480 GHz utilizada)	Banda ISM de 2,4 GHz (2,402 – 2,480 GHz utilizada)
Canais	40 canais com espaçamento de 2 MHz (3 canais de publicidade/37 canais de dados)	79 canais com espaçamento de 1 MHz
Uso do canal	Espectro de propagação de salto de frequência (FHSS)	Espectro de propagação de salto de frequência (FHSS)
Modulação	GFSK	GFSK, π/4 DQPSK, 8DPSK
Taxa de dados	LE 2M PHY: 2 Mb/s LE 1M PHY: 1 Mb/s LE Codificado PHY (S=2): 500 Kb/s LE Codificado PHY (S=8): 125 Kb/s	EDR PHY (8DPSK): 3 Mb/s EDR PHY (π/4 DQPSK): 2 Mb/s BR PHY (GFSK): 1 Mb/s
Potência Tx*	≤ 100 mW (+20 dBm)	≤ 100 mW (+20 dBm)
Sensibilidade Rx	LE 2M PHY: ≤-70 dBm LE 1M PHY: ≤-70 dBm LE codificado PHY (S=2): ≤-75 dBm LE codificado PHY (S=8): ≤-82 dBm	≤-70 dBm
Transportes de dados	Assíncrono Orientado a Conexão Isócrono Orientado a Conexão Assíncrono Sem Conexão Síncrono Sem Conexão Isócrono Sem Conexão	Orientado à conexão assíncrona Orientado à conexão síncrona
Topologias de comunicação	Malha de transmissão ponto a ponto (incluindo piconet)	Ponto a ponto (incluindo piconet)
Recursos de posicionamento	Presença: Publicidade Direção: Localização de direção (AoA/AoD) Distância: RSSI, HADM (Coming)	Nenhum

Fonte: *Bluetooth*

2.3 *Google Firebase*

De acordo com Elmasri e Navathe, banco de dados é um universo de informações relacionadas, onde os dados podem ser informações implícitas, ditas como fatos ou podem ser características que se relacionam com os fatos. A aplicação desse tipo de tecnologia abrange todos os tipos de interações com o mundo digital, que prioriza a pesquisa por padrões dessas características e com isso pode analisar e manipular resultados de forma inteligente. Essas relações inteligentes precisam ter um contexto e ser coerentes para que façam sentido com a

pesquisa de dados e para que suas análises funcionem corretamente (ELMASRI & NAVATHE, 2005).

Com essa forma de análise de dados aliada ao armazenamento em nuvem, a mineração de dados e análise de resultados para tomada de medidas produtivas se torna bastante ampla.

O *Firebase Realtime Database* é uma das inúmeras plataformas que possibilita esses modelos de análise contínuo de dados. Ela proporciona ao usuário final, além de um aplicativo mobile e responsivo mesmo offline, uma conexão segura, colaborativa e em tempo real, mostrando ao usuário qualquer tipo de alteração no banco ao reconectar o dispositivo utilizado à internet (FIREBASE, 2022).

2.4 *Android Studio*

O *Android Studio* é uma plataforma de desenvolvimento de aplicativos *Android* baseada no *IntelliJ IDEA*. Além de IDE, o *Android Studio* contém ferramentas para otimização de criação de aplicativos Baseado na linguagem Kotlin e o sistema de compilação *Gradle* (*Android Studio,* 2021).

O *Android Studio* ainda disponibiliza, assim como várias plataformas da *Google,* seu *Software Development Kit* (SDK), ferramentas fornecidas pelo fabricante de uma plataforma de hardware, sistema operacional ou linguagem de programação (RED HAT, 2020).

2.5 *Smartwatch*

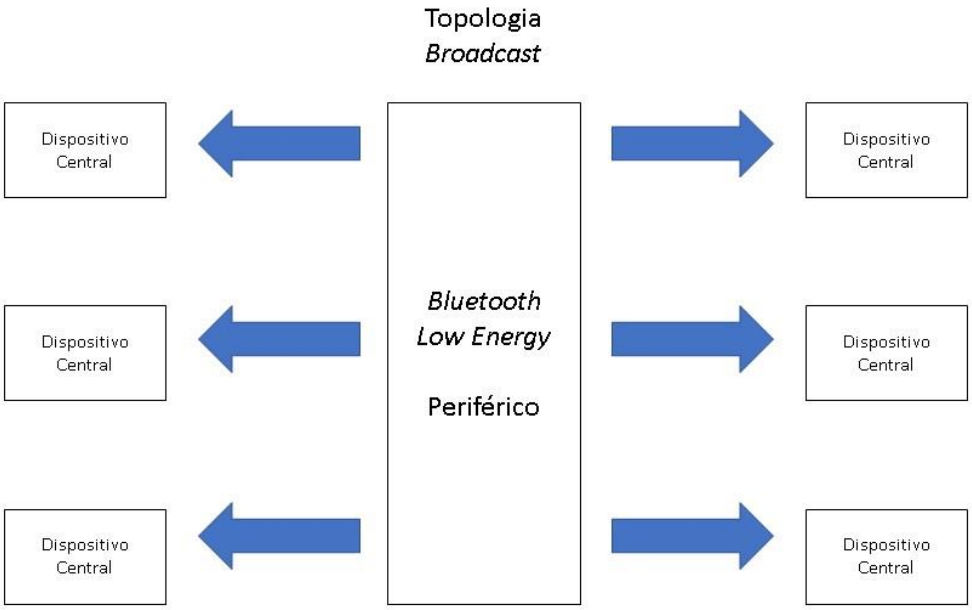
Um smartwatch nada mais é que um dispositivo *wearable*, ou dispositivo vestível, capaz de se conectar com um smartphone para trabalhar em conjunto com o mesmo. Dos mais variados exemplos de dispositivos *wearables*, tem se como populares os smartwatches, *smartbands*, óculos de realidades virtual (VR) e luvas (VALER, 2017).

Munida de sensores, implantados em suas últimas versões por diversas empresas, intimamente usados nas medições de batimento e ritmo cardíaco, mensuração de oxigênio sanguíneo e com conexão contínua à *smartphones* via *bluetooth,* são utilizados pelas pessoas como forma de análise em exercícios físicos para melhorar tempo de treino e até mesmo na dieta conforme os resultados obtidos por cada indivíduo, o que torna essa tecnologia um potencial para estudos de caso singulares a cada usuário.

2.6 Protocolos do *Bluetooth Low Energy*

De acordo com Townsend, o que torna o *Bluetooth Low Energy* (BLE) um protocolo tão utilizado e poderoso, dentre vários outros protocolos *wireless*, é sua forma inteligível de projetar algo que possa se comunicar com qualquer plataforma, tendo ela qualquer sistema operacional moderno.

Figura 4: Transmissão de um periférico para vários dispositivos centrais



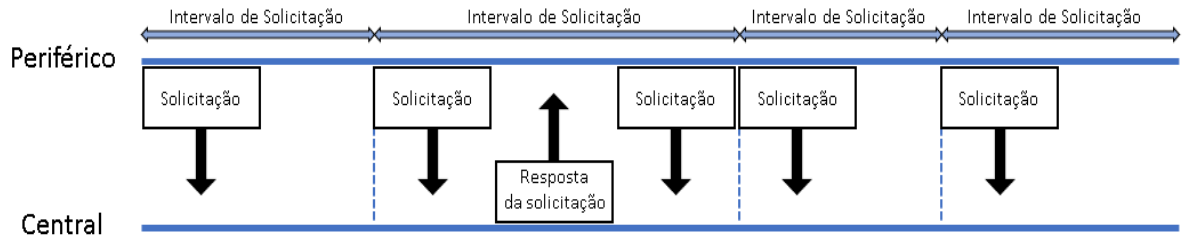
Fonte: Adaptado de Adafruit, 2014.

Para suas definições de funções de dispositivo, é utilizado o *Generic Access Profile* (GAP), determinando visibilidade ao mundo exterior e disponibilidade entre interações de dispositivos (TOWNSEND, 2014). Ainda segundo Townsend, o GAP define várias funções de dispositivos, mas dois principais conceitos devem ser essencialmente lembrados, sendo eles periféricos, com um fluxo unidirecional, do dispositivo periférico para o dispositivo central, de baixo consumo de energia que se conectam com um dispositivo mais potente, ou central, normalmente sendo tablet ou smartphones, assim visto na figura 4. O conceito de carga útil, traduzida do termo "*payload*" no texto original, é a parte real dos dados transmitidos, sendo ela a real mensagem pretendida.

Há duas formas de solicitação com o GAP: a carga útil de dados de solicitação e a carga útil de resposta de verificações idênticas, podendo conter 31 bytes de dados, sendo a carga útil de solicitação obrigatória, já que está é transmitida constantemente informando aos dispositivos

centrais, ao alcance que o dispositivo periférico existe, entende-se, portanto, que o GAP configura o cabeçalho do sistema de transmissão de dados, para que este possa ser utilizado recorrentemente (TOWNSEND, 2014).

Figura 5: Representação ilustrada das conexões GAP.



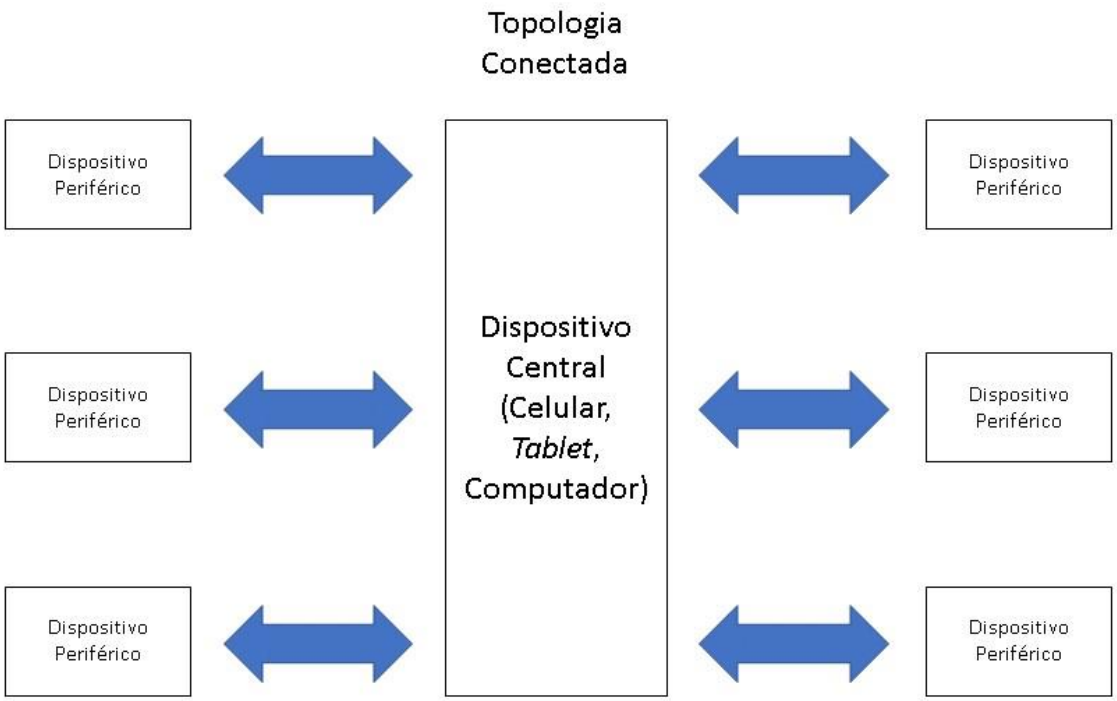
Fonte: Adaptado de Adafruit, 2014.

Um dispositivo periférico demandará um intervalo de tempo para solicitações, visto na figura 5. A cada tempo transcorrido, o periférico irá retransmitir o pacote de solicitação principal, ou seja, quanto mais intervalados são essas solicitações, menos consumo de energia, porém também será menos responsivo. Se um dispositivo de recepção tiver desejo na carga útil transmitida, que esteja disponível, ele pode mandar uma carga de resposta de varredura e será respondido com mais dados do periférico (TOWNSEND, 2014).

Após a conexão feita, o esse envio de solicitações será, normalmente, interrompido. A partir do momento em que essa conexão é feita, os serviços utilizados serão de características GATT, onde há a comunicação bidirecional.

O *Generic Attribute Profile* (GATT), ou "Perfil de Atributo Genérico", traduzido do inglês, de acordo com o Android Studio (2021), "é uma especificação geral para enviar e receber pequenas quantidades de dados como "atributos" por um vínculo do *Bluetooth Low Energy* (BLE)". Utilizando o mínimo possível de *bytes*, o GATT é um aprimoramento do protocolo de atributo (ATT), deixando-o mais otimizado para os dispositivos BLE (*ANDROID STUDIO,* 2021).

Figura 6: Conexão bidirecional do GATT.

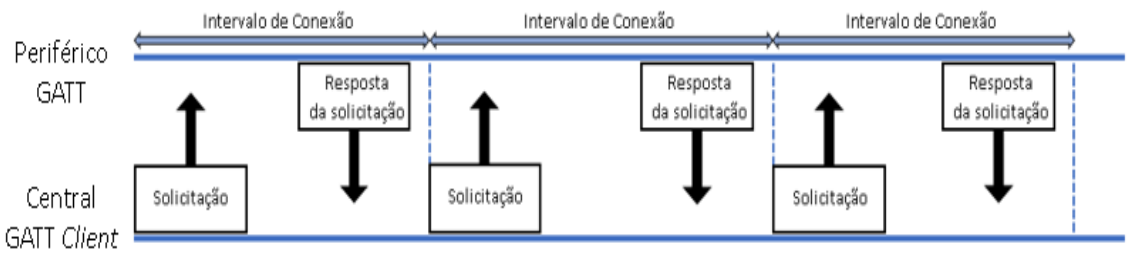


Fonte: Adaptado de Adafruit, 2014.

Como qualquer outro protocolo, é necessária a definição das funções de cliente e servidor. O cliente GATT se comunica com o servidor enviando requisições e recebendo respostas. Não há conhecimento do servidor e de seus atributos antes da conexão, portanto é necessária a requisição por parte do cliente de antemão. Após a descoberta é possivel a leitura e gravação no servidor (DAVIDSON, 2022).

Ainda segundo Davidson, o Servidor GATT recebe e retorna mensagens quando solicitado, podendo ainda atribuir atualizações quando programado para tal e define o que irá armazenar ou disponibilizar informações ao cliente. Cada dispositivo BLE deve ter um servidor GATT básico para responder às solicitações do cliente.

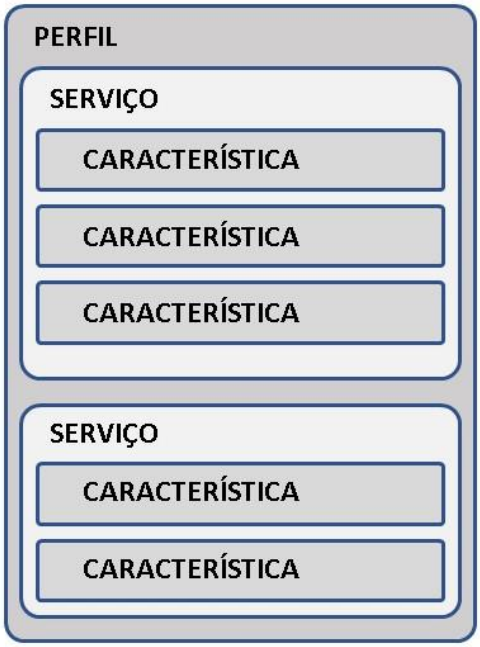
Figura 7: Conexão intervalada do GATT



Fonte: Adaptado de Adafruit, 2014.

O GATT é baseado em 3 âmbitos que englobam entre si, sendo eles perfil, serviços e características, ilustrado abaixo na figura 8:

Figura 8: Características do GATT.



Fonte: Adaptado de Adafruit, 2014.

2.6.1 Perfis

Os *Profiles*, ou "Perfis" não existem realmente nos periféricos BLE, eles são simplesmente uma coleção de serviços predefinidos compilada pelo designer do periférico.

2.6.2 Serviços

Os serviços são divididos em entidades lógicas e características, como é visto na figura 8, que são dados específicos. Os serviços se distinguem um do outro por meio de um identificador numérico único chamado UUID (Identificador Único Universal), que pode ser de 16 bits, sendo esses oficiais, ou 128 bits, que são serviços personalizados. Os serviços oficiais adotados pelo BLE se encontram no site da *Bluetooth Developer Portal* (*Bluetooth, 2022*).

2.6.3 Características

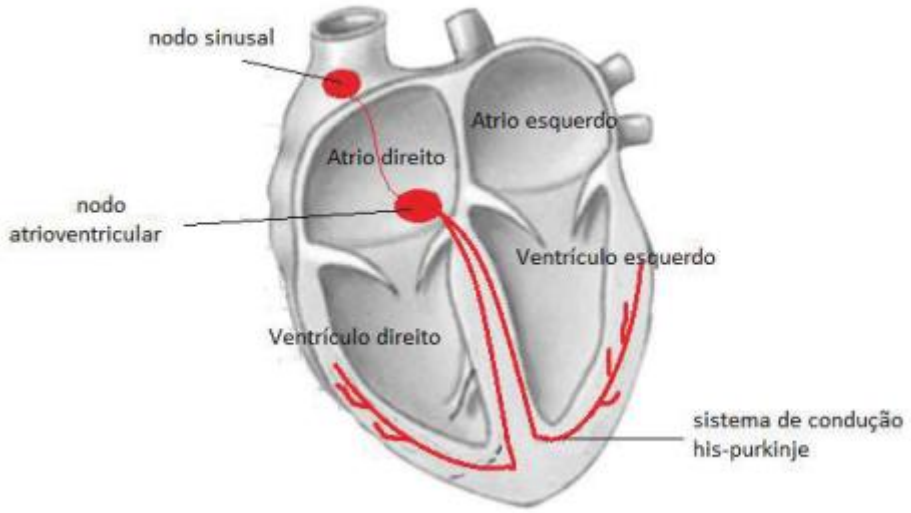
O mais baixo nível do nas transações GATT são as características, nelas são encapsulados pontos únicos de dados, mas podem conter dados relacionados, como coordenadas (*Bluetooth, 2022*).

Segundo *Bluetooth,* as características são similares aos serviços, já que cada característica distingue entre si via um UUID predefinido de 16 ou 128 bits, e são o ponto principal para interação com um periférico.

2.7 Arritmias Cardíacas

Segundo a SOBRAC (Sociedade Brasileira de Arritmias Cardíacas), o coração tem dois átrios e dois ventrículos, que são duas cavidades na câmara superior e duas cavidades na câmara inferior, respectivamente. Seu bombeamento, que leva o oxigênio pelo corpo todo é feito por um sistema elétrico conhecido como nó sinusal ou sinoatrial, localizado no átrio direito (evidenciado na figura 9), gerado a partir de células que o efetuam de forma contínua e ininterrupta para o átrio contrário (SOBRAC, 2018). A SOBRAC ainda evidencia que, quando o nó sinusal tem má formação ou o coração tem má condução do impulso elétrico, acontecem as arritmias que podem ser nomeadas como taquicardia, quando a frequência cardíaca é maior que a frequência padrão do paciente braquicardia, quando a frequência cardíaca é menor que a frequência padrão do paciente, além da fibrilação, que é o nome dado à frequência irregular em relação à frequência padrão.

Figura 9: Anatomia do coração com relação ao nó sinusal (nodo sinusal).



Fonte: PEREIRA, 2018.

O presidente da SOBRAC, Dr. José Carlos Moura, integra informando que indicar sintomas para que as pessoas busquem ajuda a tempo é essencial para reverter as estatísticas de mortes e melhor a qualidade de vida em torno das arritmias. Dr. Moura finaliza dizendo: “Junto destes sintomas, a pessoa também pode realizar um teste simples de medição do próprio pulso ao longo do dia para conferir os batimentos cardíacos, que devem estar entre 50bpm e 100bpm quando a pessoa está em repouso”.

2.8 Trabalhos relacionados

Nesse tópico serão mostrados trabalhos que tangenciam o tema de forma a agregá-los ao projeto atual.

2.8.1 *Uso de SmartWatch no Auxílio a Monitoração de Arritmias Cardíacas (2020)*

O trabalho em questão teve como objetivo fazer um aplicativo de registro de paciente e cuidador, onde o paciente utilizava o smartwatch e aferia automaticamente ou manualmente seus batimentos cardíacos e a partir dos resultados seria informado ou não ao cuidador (SILVA; FERNANDES; LINS, 2020).

O aplicativo tem a disposição também de um banco de dados mostrando o histórico de medições de batimentos cardíacos, GPS para localização e socorro do paciente pelo cuidador e

alternância de atividade para melhor medição de batimentos (SILVA; FERNANDES; LINS, 2020).

O trabalho concluiu que o projeto auxiliaria o cardiologista do paciente conforme as medições preservadas no histórico ao mesmo tempo que auxilia o salvamento do paciente anexado ao cuidador (SILVA; FERNANDES; LINS, 2020).

2.8.2 *SafeWatch: Detectando quedas com Smartwatches*

O trabalho aqui citado tem como objetivo a criação de um aplicativo para estabelecer um cálculo afim de reconhecer o momento em que o usuário com o *SmartWatch* tem uma queda e a partir disso enviar uma notificação, questionando se o usuário necessita de auxílio (FARIA, TAVARES & VIEIRA, 2017).

O projeto tem também auxílio do GPS para verificação da geolocalização do usuário que necessita de ajuda (FARIA, TAVARES & VIEIRA, 2017).

3 PROPOSTA DE SOLUÇÃO

Este capítulo apresenta a proposta de solução, complementando com os trabalhos anteriores a possível proposta de utilização de *smartwatch* com utilização de seus sensores em comunicação com *smartphone* via *bluetooth low energy.*

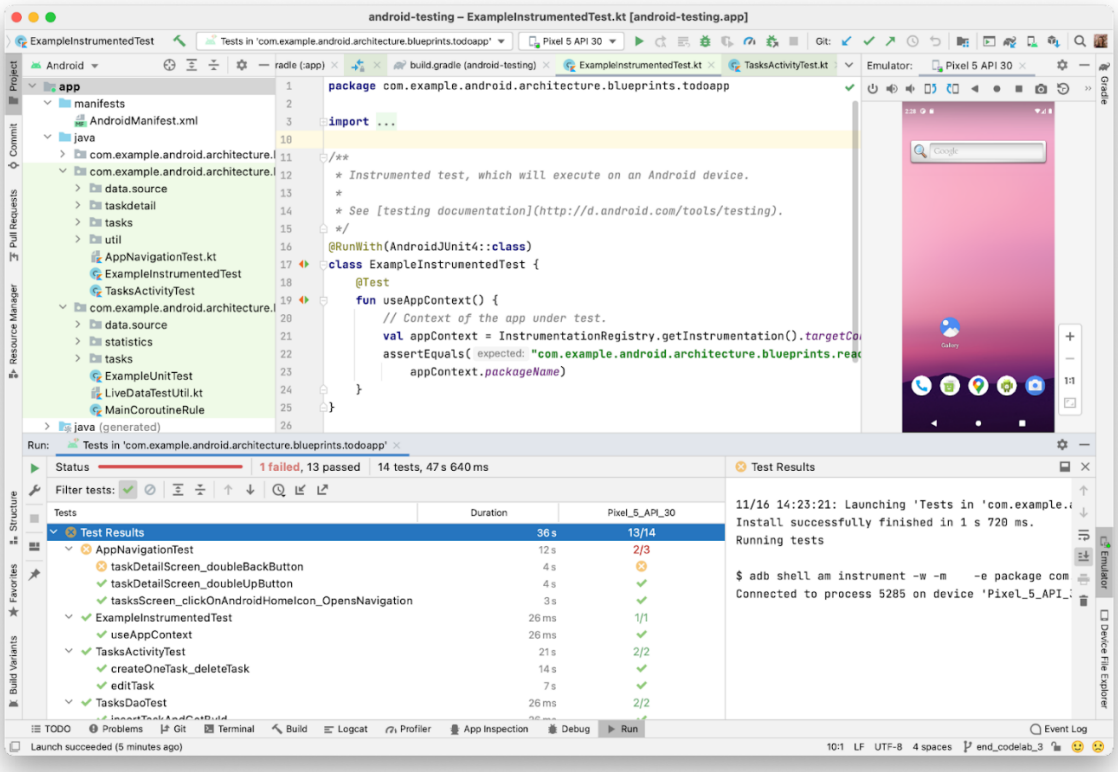
3.1 Componentes de *Software*

A presente seção descreve as ferramentas de software utilizadas na prototipagem do trabalho.

3.1.1 *Android Studio*

Podendo ser integrado no *Mac*, *Windows* ou *Linux*, o *Android Studio* é um IDE (do inglês *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) que permite um desenvolvimento integrado de uma aplicação do sistema *Android* (*ANDROID STUDIO*, 2022). O ambiente tem em sua coletânea de ferramentas, um emulador de dispositivos *Android,* que além de emular, simula o dispositivo. É possível também fazer edições de telas, utilizando elementos de arraste a IU (interface de usuário), do editor de *layout* visual (Figura 9) e ainda blocos de programação inteligentes que podem ser utilizados no editor de código (*ANDROID STUDIO*, 2022).

Figura 10: Editor de *Layout Visual*



Fonte: *Android Studio, 2022*

Os aplicativos *Android* mantêm quatro blocos de construção, que são chamados componentes. Cada bloco é um meio de entrada do sistema ou usuário no aplicativo, sendo eles: atividades, serviços, *broadcast receivers* e provedores de conteúdo. Cada tipo de bloco de construção mantém sua finalidade e ciclo de vida distintos, definindo como o componente se comporta dependendo de como ele é utilizado. (*ANDROID STUDIO, 2022*).

A atividade é o meio de interação inicial do usuário, com um aplicativo. Há uma tela da interface para cada atividade, sendo elas independentes, mesmo que se em conjunto elas formem uma forma coesa de experiencia para o usuário. Para o aplicativo manter sua execução é necessário o serviço, sendo este um componente de execução longa ou para processos remotos, não apresentando interface ao usuário. Os *broadcasts receivers* entregam eventos a aplicativos fora de fluxo de usuários comum, permitindo que o aplicativo responda às solicitações de transmissão pelo sistema inteiro, entregando transmissões até para aplicativos que não estão em execução. E, por fim, os provedores de conteúdo gerenciam dados de aplicativo, podendo armazená-los em um sistema de arquivos, um banco de dados *SQLite*, na

Web ou em qualquer lugar de armazenamento persistente acessível, permitindo que outros aplicativos os leiam ou modifiquem (*ANDROID STUDIO, 2022*).

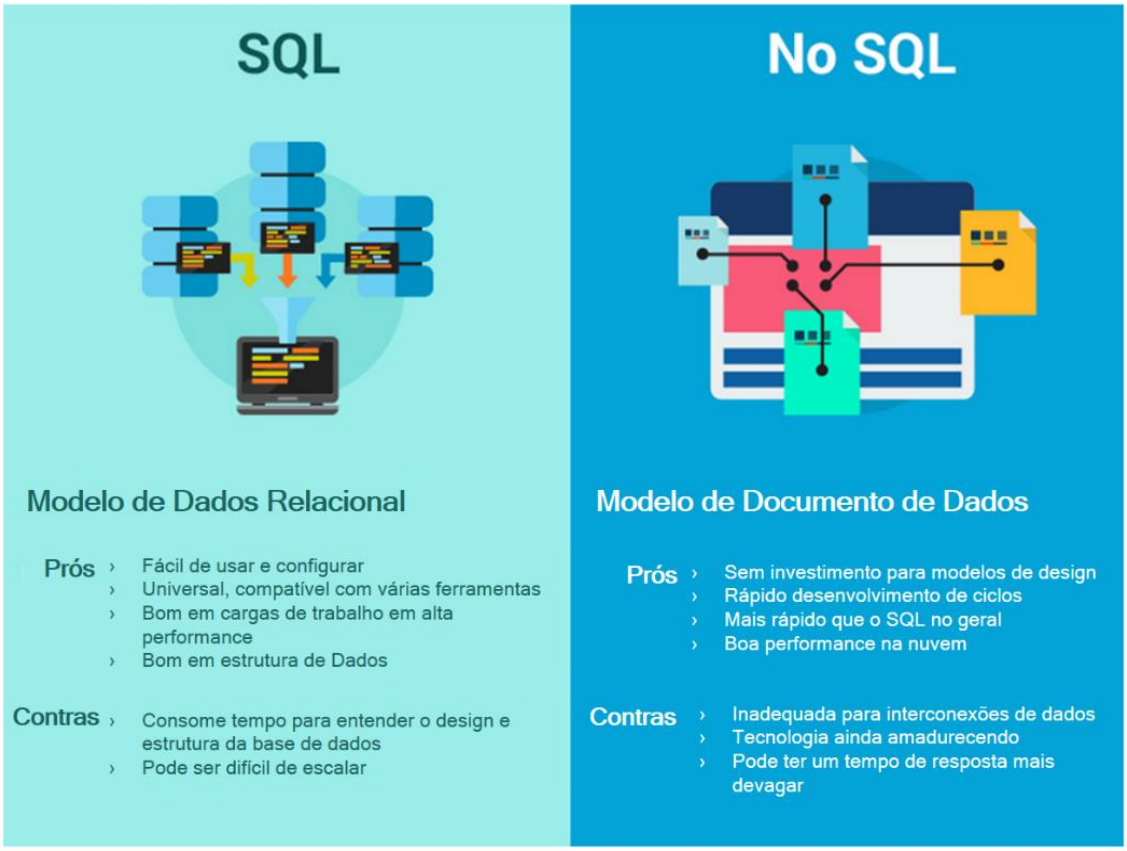
Com base nessas qualidades e no intuito do projeto trabalhado, o *Android Studio* está sendo amplamente comentado e utilizado, mantendo o esforço nas telas dos aplicativos e editor de código que serão utilizados, tanto para a programação e emulação do aplicativo *mobile* para *smartphone*, conforme este trabalho evolui.

3.1.2 *Google Firebase*

O *Google Firebase Database* se hospeda em nuvem armazenada em arquivos JSON e sincronizada em tempo real para todos os clientes ambientados, utilizando sincronização de dados às solicitações HTTP (*Hypertext Transfer Protocol* ou Protocolo de Transferência de Hipertexto em português) típicas. Além de SDKs (*Software Development Kit* ou *Kit* de Desenvolvimento de *Software* em português) para *Android*, a plataforma ainda contém kits de desenvolvimento para *Apple* e *JavaScript*, os quais mantêm os aplicativos responsivos mesmo offline, já que armazena os dados em disco, mantendo a experiência responsiva e atualizando alterações feitas assim que conectado (FIREBASE, 2022).

O *Firebase,* sendo uma opção *NoSQL*, apresenta uma variação de otimizações e funcionalidades dos bancos de dados relacionais (Figura 10). Também permite uma criação segura de aplicativos avançados e colaborativos com a segurança de seus usuários, concedendo permissão identificada para quais dados são acessados e quais clientes podem acessá-los, integração de seu sistema próprio, chamado *Firebase Authentication.* (*FIREBASE*, 2022).

Figura 11: Diferenciação de tipos de base de dados



Fonte: *Clarion Technologies*

O *Firebase* está sendo utilizado nesse trabalho para armazenamento de dados dos usuários como forma de ampliação de medidas de sinais vitais, para padronização e possível personalização.

3.2 Componentes de Hardware

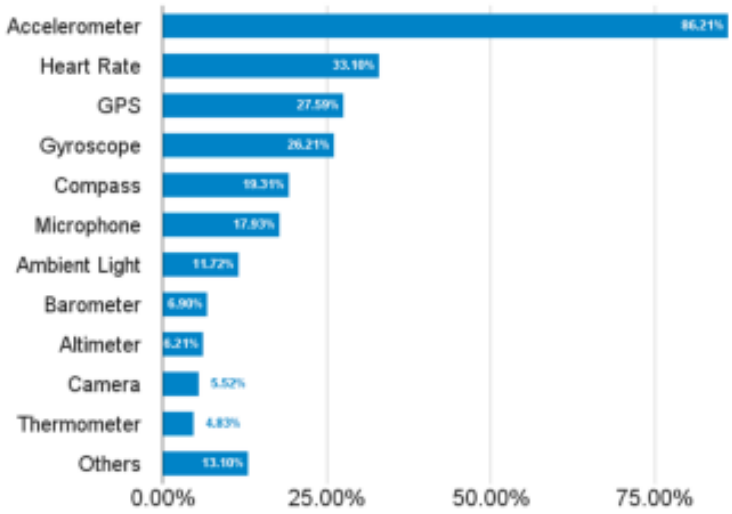
Esta seção tem como função ampliar os conceitos de hardware que será utilizado no projeto.

3.2.1 SmartWatch

O *smartwatch* foi concebido com o objetivo inicial de auxílio em cálculos, sendo sua primeira versão criada na década de 80, contendo uma calculadora embutida, desenvolvida pela Casio, evoluindo até os dias de hoje, onde se tem conexão com o *smartphone* via *bluetooth* e reproduzindo quase todas as funcionalidades do mesmo (PEDROS, ABREU, VIEGAS, 2016).

Juntamente com a variedade de SOs e plataformas, foi incluída também uma variedade de sensores nos wearables, ou vestíveis. Na figura 11, é possível analisar os sensores encontrados nos mais de 140 vestíveis examinados, sendo os mais comuns os sensores relacionados a movimento e frequência cardíaca (PÉREZ; RODRÍGUEZ; GAGO, 2016)

Figura 12: Sensores disponíveis nos vestíveis.



Fonte: PÉREZ; RODRÍGUEZ; GAGO, 2016

Este trabalho está utilizando os sensores de um *smartwatch* para análise e supervisão de dados para ocorrências de emergências e alertas de situações personalizadas com base em cada usuário do dispositivo, nesse projeto está sendo utilizado o sensor de frequência cardíaca.

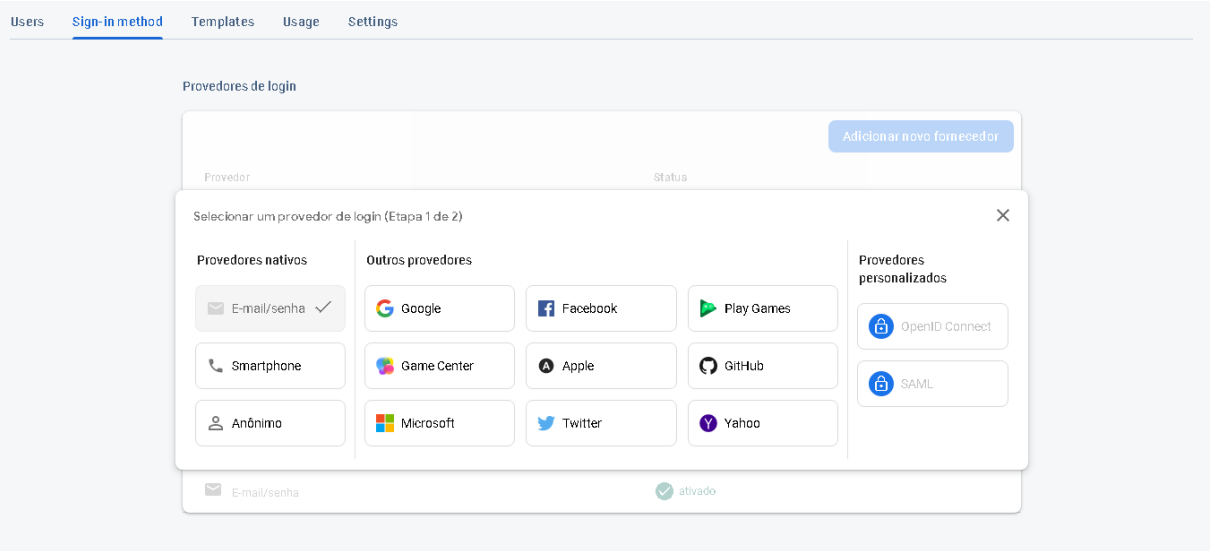
4 Componentes de construção do aplicativo

Neste capítulo serão citadas as ferramentas para a construção do software e suas especificações, para possíveis ajustes futuros e documentação.

4.1 *Firebase Authentication*

Os *SDKs* do *Firebase Authentication* permitem um processo simples de implementação do processo de autenticação de usuários utilizando de vários meios de autenticação possível, podendo utilizar de contas do *Google*, *Facebook*, *Twitter*, *Github* e outros, podendo ser observados na figura 13, fornecendo métodos para gerenciar usuários, lidando também com envios de e-mails e redefinição de senhas (*FIREBASE*, 2022).

Figura 13: Tela de inclusão de métodos de autenticação no banco de dados.



Fonte: *Firebase,* 2022.

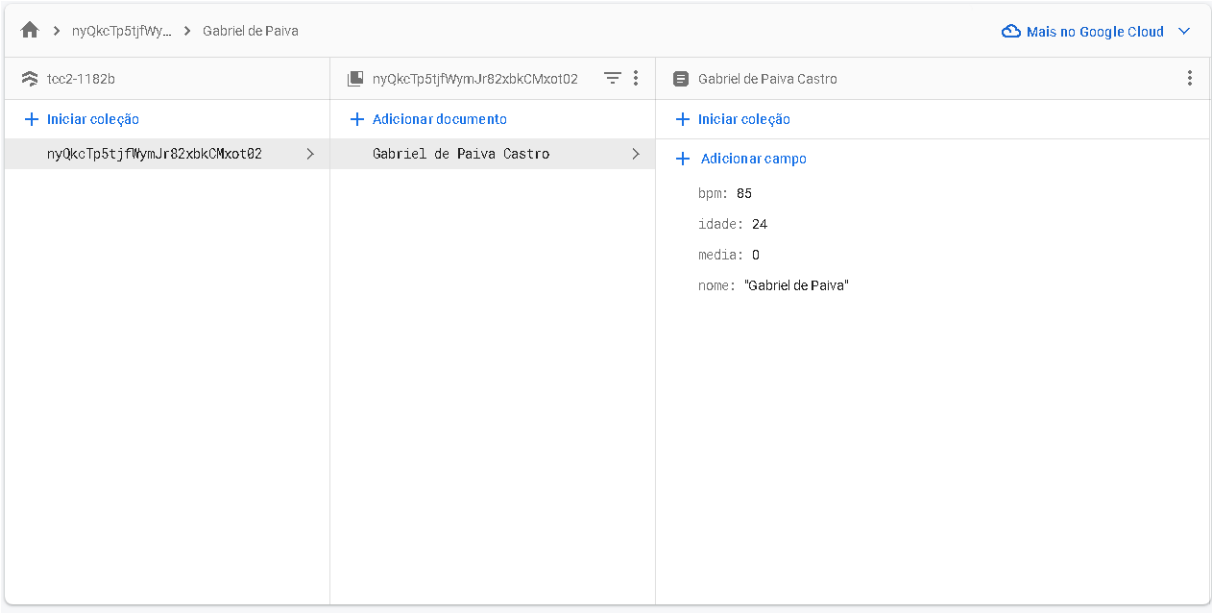
Para a implementação do banco de dados foi utilizado o *Cloud Firestore,* que mantém métodos de implementação e incrementos de dados de uma forma simples e flexível para uma melhor modelagem e sincronia para aplicativos clientes, utilizando ouvintes de atualização de dados permitindo alterações em tempo real para todos os usuários conectados (*FIREBASE*, 2022).

No modelo de dados *NoSQL* do *Cloud Firestore*, é possível fazer o armazenamento de dados com um mapeamento de dados compostos por campos e valores, sendo o armazenamento mantido em coleções e containers de documentos, mantendo assim vários níveis de hierarquia,

podendo suportar vários tipos de dados diferentes em um mesmo documento e coleção (*FIREBASE*, 2022).

Com isso a utilização do *Firestore* foi essencial no processo de guardar e referenciar dados dos usuários que utilizaram o *APP*. No intuito de monitoramento de pacientes por meio de um aplicativo de celular, o método utilizado consiste em cadastrar o paciente, pelo nome, sobrenome, idade, média de batimentos cardíacos e frequência de batimentos por minutos, utilizando do nome e sobrenome juntos como o nome de acesso documento do paciente, assim como na figura 14.

Figura 14: Exemplo de dados vistos do painel do *Google Firebase* criados pelo *APP*



Fonte: Elaborado pelo autor

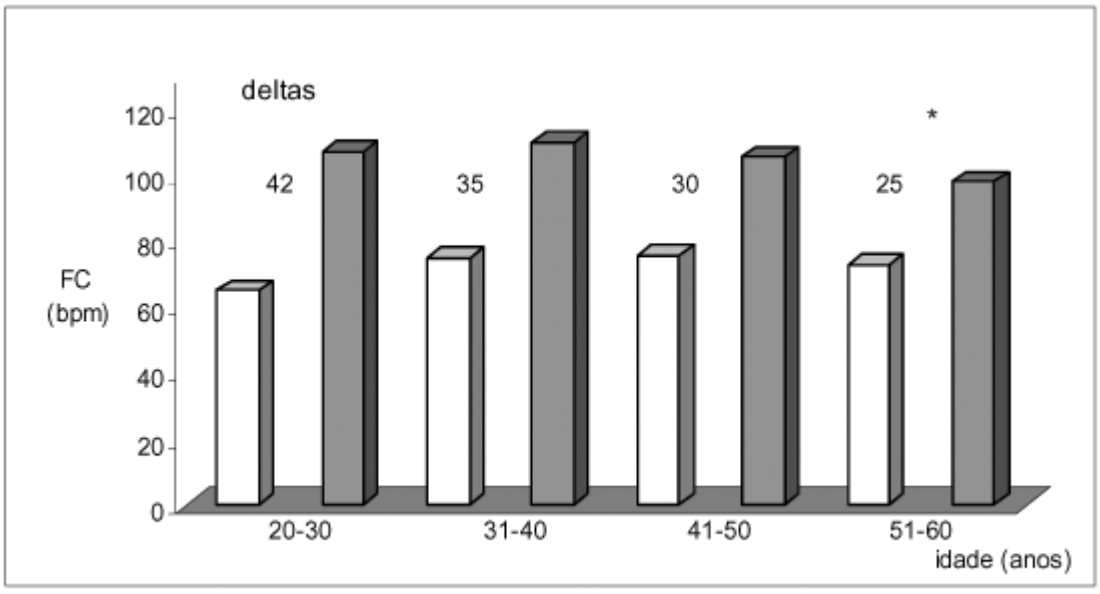
Os valores de média e bpm, podem ser colocados manualmente, ou com as medidas proporcionadas pelo aparelho medidor, que neste trabalho apresentado constitui no aparelho *smartwatch* genérico do modelo *Hero Band B57*, mostrado na figura 15, a seguir.

Figura 15: *Smartwatch* utilizado nas medições de batimentos cardíacos do projeto

Fonte: *Amazon*, 2022.

É verificado com o usuário ou com seu cuidador se a média escolhida será utilizada por meio da inserção do próprio ou por meio de medições. O projeto foi implementado em relação à média, baseado no estudo de Paschoal, feito com 40 pessoas que fez a verificação das frequências cardíacas de pessoas em várias faixas etárias na posição supina (no gráfico, na cor branca) e bípede (no gráfico, na cor cinza), mostrando que há certa relação referente a idade do paciente e sua frequência cardíaca assim como sua posição corporal no momento da aferição, como visto na figura 16 (Paschoal MA, 2006).

Figura 16: Valores medianos de FC para cada grupo e faixa etária



FC = freqüência cardíaca; *p<0,05 grupo mais velho vs demais grupos

Fonte: Paschoal MA, 2006.

Tendo isso em foco, o projeto também faz questão de coletar a idade do paciente, podendo assim coletar esse dado para futuros projetos de análise de dados.

4.2 Atividades

A *activity* ou atividade é essencial para um aplicativo *Android*, tendo como prioridade de planejamento o modo como suas atividades são distribuídas e reunidas, utilizando de sua instância *Activity,* invocando seus métodos de *callback* adequados aos seus estágios de ciclo de vida, entre os mais usuais, estão: *onCreate*, correspondente ao estágio acionado à criação da atividade; *onStart*, correspondente ao momento de início da atividade; *onPause*, corresponde ao momento que o usuário sai da aplicação pelos métodos de "voltar" ou "mais recente", *onStop*, correspondente ao momento que a aplicação não é mais vista pelo usuário e *onDestroy*, correspondente ao estágio de destruição da atividade (*ANDROID DEVELOPERS*, 2022).

4.2.1 Atividades de *Login* e Cadastro

As atividades principais, login e cadastro se baseiam nas atividades citadas anteriormente e conforme são adicionados ao projeto, geram dois arquivos de estrutura de código:

- *activity_login.xml* e *activity_cadastro.xml*: estes arquivos são responsáveis pela geração das interfaces e da área de visualização do usuário (*ANDROID DEVELOPERS,* 2022);
- *LoginActivity.kt* e *CadastroActivity.kt*: estes arquivos contém os comandos de intercomunicação entre os arquivos de interface e a parte de construção do funcionamento do aplicativo (*ANDROID DEVELOPERS*, 2022).

Neste caso em específico, são responsáveis pelo cadastro e acesso ao banco de dados por meio de um e-mail e senha, a partir deles é possível fazer o acesso à atividade principal, onde é possível fazer acesso ao fragmento de monitoramento, adição de paciente/usuário, medição de frequência cardíaca para atualização de medidas. A adição, atualização, leitura e remoção de coleções de pacientes no banco de dados *firestore* só é possível a partir do login no mesmo, onde ao fazer o login, cada usuário recebe um id específico, tornando o acesso de seus pacientes cadastrados vinculado ao seu id único gerado pelo *Firestore*, sem maneiras de acesso de outros usuários.

O primeiro contato com o *APP* feito para a implementação do projeto é feito por meio de um *login,* contendo a apresentação inicial do app, feito a partir do *Firebase Authentication*, no qual é possível utilizar de um e-mail válido e uma senha de ao menos 6 dígitos para que seja possível entrar na tela inicial do projeto visto na figura 17, em caso de não houver um cadastro válido é possível acessar a tela de cadastro por meio desta primeira tela e registrar um e-mail e senha, visto na figura 18, o projeto aqui apresentado utilizou apenas dos métodos de autenticação de e-mail e senha. Foi utilizado de medidas protetivas para verificação de qualidade de senha para a aplicação do cadastro da conta, assim como verificação de rede e de possível conta duplicada, padronizadas pelo SDK do Firebase, tornando a implementação simplificada.

Figura 17: Tela do protótipo do projeto de *Login* para acesso ao banco de dados e *APP*

Fonte: Interface do dispositivo de monitoramento, elaborado pelo autor.

No momento do registro ou do login, o usuário se torna o usuário da instância atual, mantendo seu estado. Sendo assim, atualizações na página ou no navegador não permitem perda de dados. Da mesma forma, quando o usuário fecha o *APP*, sua instância continua funcional, não perdendo os dados de login quando retornar (*FIREBASE*, 2022).

A partir do momento inicial, onde o usuário faz a criação de sua forma de autenticação e faz *login*, o projeto foi implementado utilizando uma forma de manter os dados no dispositivo

para que da próxima vez que o mesmo usuário for utilizar o *APP*, já faça o login instantâneo sem a necessidade de uma segunda autenticação, mantendo funcional a referência ao último *login*.

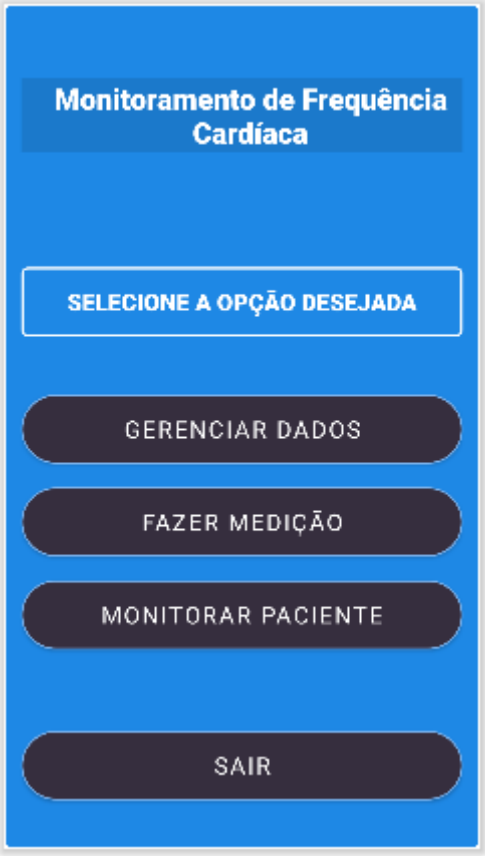
Figura 18: Tela do protótipo do projeto de cadastro para acesso à tela de *Login*

Fonte: Interface do dispositivo de monitoramento, elaborado pelo autor.

4.2.2 Atividade Principal

A atividade Principal, será o menu base para "inflar" os fragmentos que serão selecionados no menu de botões central da tela do fragmento *Option*, iniciando o aplicativo, após o cadastro e o login, na área principal do projeto, permitindo ao usuário que faça o gerenciamento de pacientes, no botão "Gerenciar Dados", fazer medição utilizando o dispositivo smartwatch via bluetooth, no campo "Fazer Medição" e por fim acompanhar os dados de algum paciente específico pelo campo "Monitorar Paciente", além de poder sair para a tela de login, no botão "Sair", como visto na figura 19.

Figura 19: Menu de escolhas de fragmentos do *APP*



Fonte: Interface do dispositivo de monitoramento, elaborado pelo autor.

4.3 Fragmentos

Um *fragment* ou fragmento forma uma parte de integração da interface do usuário em uma *FragmentActivity,* podendo combinar vários desses campos de fragmentos em uma única atividade, criando assim um *IU* (*User Interface,* ou interface de usuário) com vários tipos de painéis modulares, sendo eles tipos de subatividades, tendo seus próprios ciclos de vida, além de ter suas próprias entradas, tendo a capacidade de serem executados ao mesmo tempo que a atividade principal (*ANDROID DEVELOPERS*, 2022).

O fragmento deve ter uma atividade como hospedagem, tendo o ciclo de vida do mesmo um grande impacto no ciclo de vida da atividade do *host*, ou seja, enquanto a atividade está em execução, ainda é possível processar cada fragmento independentemente, adicionando-os e removendo-os, porém se a atividade é destruída, os fragmentos também o são (*ANDROID DEVELOPERS*, 2022).

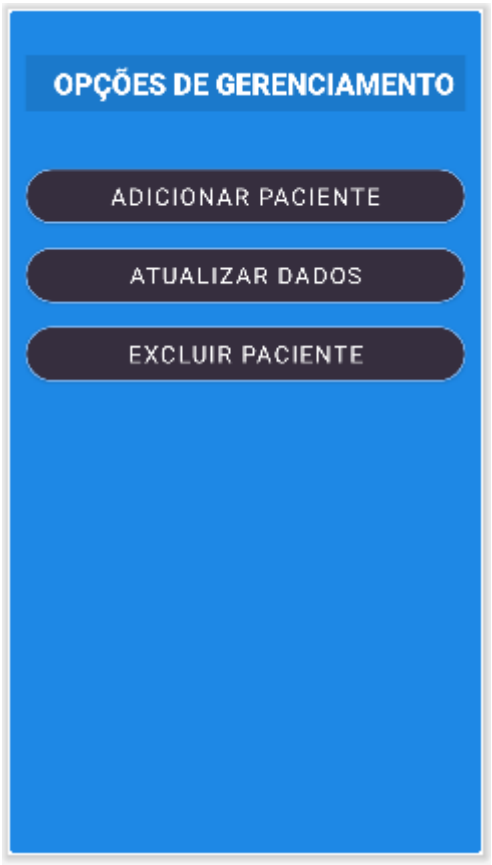
Estes fragmentos admitem que projetos de tablets e layouts com telas maiores, tenham aplicações com maior flexibilidade e sem a necessidade de gerenciamento de alterações complexas na hierarquia de visualizações (*ANDROID DEVELOPERS*, 2022).

4.3.1 Fragmentos de Gerenciamento

No protótipo deste projeto, ao efetuar o *login*, o paciente ou o cuidador pode efetuar cadastro de seus pacientes dentro do menu de gerenciamento (acessado no botão "Gerenciar Dados" visto na figura 15), no botão "Adicionar paciente", acessando o fragmento de gerenciamento, mantendo assim um documento com os dados do paciente monitorado no *Cloud Firebase*. Caso já exista um paciente cadastrado, é possível atualizar seus dados pelo método de atualização em "Atualizar dados" no menu e excluí-lo pelo método de exclusão em "Excluir paciente", encontrado no menu de opções de gerenciamento da figura 20.

Neste fragmento se encontra a função trocarFragmento(), sendo está a tela principal após o *login* e o cadastro, para que o cuidador ou paciente.

Figura 20: Protótipo da tela de gerenciamento de paciente do *APP*



Fonte: Interface do dispositivo de monitoramento, elaborado pelo autor.

Na tela de cadastro é possível, além de cadastrar o paciente com sua idade, cadastrar uma média padrão de batimentos e uma frequência cardíaca que possam ter sido medidos externamente, anteriores ao uso do aplicativo ou podendo ser medidos na área de mensuração do aplicativo, mantendo a robustez do monitoramento, que é a prioridade no aplicativo deste trabalho.

Ao clicar no “Adicionar Paciente”, o app leva até o fragmento de acesso ao cadastro do paciente, podendo assim colocar os dados indicados e salvá-los para o banco de dados, visto na figura 21.

Figura 21: Protótipo da tela de cadastro de paciente do *APP*

Fonte: Interface do dispositivo de monitoramento, elaborado pelo autor.

Para efetuar o cadastro de um paciente, o dispositivo pega o identificador associado ao e-mail no qual foi feito o login e cria uma pasta de colecionáveis, com esse identificador como nome desta. Ao clicar no botão cadastrar paciente, ele faz um acesso à esta pasta, identificando se nessa coleção há uma pasta com o nome do paciente, no qual é salvo seus dados. Caso já existe, é mostrado um *pop-up* de aviso evidenciando a existência de um paciente com o mesmo nome, deixando o usuário com o papel de definir a identificação de cada paciente, caso haja duas pessoas com o mesmo nome. Para a adição de um novo usuário foi criado um construtor

do objeto “Usuário” com os dados de implementação idade, média e bpm (batimentos por minuto), sendo seus parâmetros.

Caso não exista, a pasta com os dados do paciente é criada, implementando os dados que foram digitados nos campos da figura 21, tendo ainda um alerta para afirmar a inclusão deste paciente, pelo método do *AlertDialog*, no qual uma mensagem aparece na tela, com botões de afirmação ou cancelamento da ação que está sendo feita.

Além deste fragmento, no caso de querer alterar algum dado de um paciente já existente no banco de dados, há também o fragmento de atualização de dados do paciente.

O procedimento para atualização se assemelha ao de cadastro, já que, para fazer a atualização de um paciente, é necessário verificar se o mesmo já foi cadastrado no banco de dados, logo é feito um acesso de *listener* ao apertar o botão “Verificar”e caso haja um paciente com o mesmo nome, na mesma coleção em relação ao nome digitado no fragmento de atualização, visto na figura 22

O processo de atualização é feito pegando os dados do banco, organizando-os segundo seus campos no fragmento, e permitindo sua alteração nos mesmos. Assim, ao apertar em “Atualizar dados do paciente”, os campos com os valores alterados são salvos na nuvem, dentro da pasta do usuário selecionado e verificado, alterando seus dados antigos.

Figura 22: Protótipo de tela de atualização de paciente do *APP*

Fonte: Interface do dispositivo de monitoramento, elaborado pelo autor.

Em caso de querer excluir um paciente do âmbito da coleção, foi criado um fragmento de exclusão de dados do paciente, visto na figura 22.

Figura 23: Protótipo de tela de exclusão de paciente do *APP*

Fonte: Interface do dispositivo de monitoramento, elaborado pelo autor.

Nesse fragmento, novamente, é verificado se o paciente existe na pasta integrada ao login do usuário. Não é possível excluir um paciente diferente do digitado, tornando um ambiente mais seguro de dados. Em caso de existir o paciente, ao apertar o botão de "Excluir dados do paciente" o usuário deve confirmar a exclusão do mesmo. Ao ser em excluídos, os campos de dados são deletados na nuvem, juntamente com a pasta com o nome do paciente, para que não fique resquícios de dados que podem prejudicar outras funções, como adição de paciente e atualização, em caso de nomes idênticos.

Em todos os casos de alteração e adição de pacientes, foram feitas verificações ao banco de dados e ao apertar para salvar os dados alterados, em todos os casos há uma confirmação de modificação de dados.

4.3.2 Fragmento de medição de frequência cardíaca

Para efetuar a medição dos batimentos cardíacos, é necessário pareamento com o dispositivo que irá obter a medida e seu nome de dispositivo para identificação na tela, utilizando assim, do sistema de conexão *GATT* que é feito utilizando o *Bluetooth Low Energy*, que se assemelha a um sistema de servidor (*mobile*) e cliente (*smartwatch* ou qualquer monitor

de sistemas vitais). Assim, o serviço do sistema de pareamento é executado, obtendo o resultado do medidor de batimentos do aparelho de monitoramento, visto na figura 24, finalizando com o salvamento na base de dados da nuvem do paciente verificado na mesma tela, integrando o sistema de monitoramento físico com o banco de dados da nuvem.

Além disso, foi implementado um *switch*, o qual, caso esteja ligado, faz uma média aritmética dinâmica da frequência dos batimentos mensurados, salvando-os na nuvem, ou dentro da pasta do paciente requisitados neste mesmo fragmento. No caso de estar desligado, a média se manterá a mesma salva no momento de cadastro do paciente. Este conceito foi integrado para que haja um método de fazer a média de batimentos, sem a necessidade desse dado anteriormente.

Figura 24: Protótipo de tela de medição de frequência cardíaca por monitor *BLE*

Fonte: Interface do dispositivo de monitoramento, elaborado pelo autor.

Após o cadastro e a efetuação das medidas pelo aparelho de monitoramento de frequência cardíaca, *smartwatch*, os dados do batimento cardíaco são enviados para o *Firebase* e salvos nos arquivos do paciente, podendo assim, ser monitorados em tempo real, por um cuidador ou médico, com o mesmo aplicativo, na tela alcançada ao clicar em “monitorar paciente”, como visto na figura 25, abaixo.

Figura 25: Protótipo de tela de monitoramento de paciente

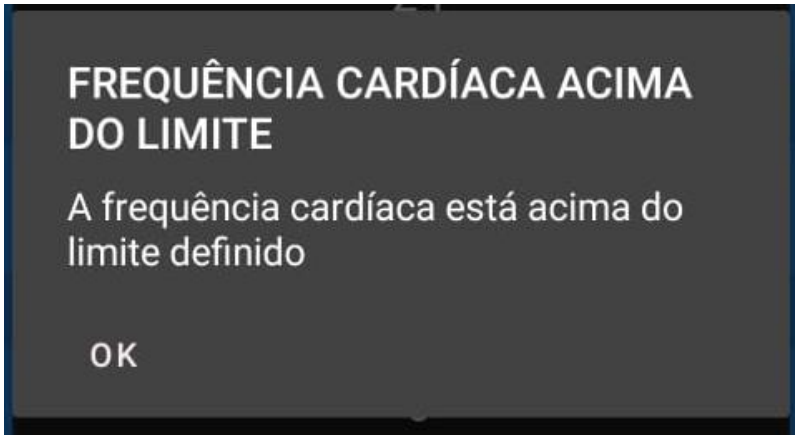
Fonte: Interface do dispositivo de monitoramento, elaborado pelo autor.

O *APP* se apoia no sistema de escutáveis ou *listeners*, que consegue emitir as mudanças relacionadas aos dados em tempo real do paciente cadastrado. No sistema mostrado na figura 25, basta digitar o nome do paciente no qual é preciso fazer o monitoramento, e após apertar o botão de monitorar paciente, abaixo do nome e sobrenome, os dados informados a baixo serão atualizados automaticamente, sem mais nenhum toque de tela necessários, permitindo assim uma medida de batimentos cardíacos com um monitoramento também contínuo, resultando em um monitoramento com um orçamento mais baixo, mas que permite um auxílio e conforto maior ao paciente e o seu cuidador, permitindo uma maior distância de monitoramento necessário.

Ainda nessa tela, é possível implementar limites que são adicionados à média, mantendo uma possível margem de erro, acionando assim, uma notificação ao celular de quem monitora, em caso de ultrapassar, tanto acima do limite quanto abaixo do limite, podendo manter um certo nível de emergência, para melhor monitoramento com mais flexibilidade ao paciente monitorado e maior dinâmica nos limites de sua frequência cardíaca.

Foi mantida a personalização dos limites na tela, já que esse tipo de margem de erro, não é padrão para todo tipo de pessoa, podendo alterar conforme idade, níveis de ansiedade, movimento do corpo e certas doenças crônicas e genéticas. Além disso, o cuidador que monitora o paciente está livre para fazer alterações nesse tipo de delimitadores, conforme o paciente altera algum tipo de exercício, mantendo o *APP* mais flexível e personalizado mostrando ainda um alerta para quando a frequência cardíaca passa dos limites informados, por meio do *DialogAlert*. Caso os limites não sejam informados, o switch no meio pode ser desligado, para que não seja gerado alertas como visto na figura 26, abaixo.

Figura 26: Alerta de frequência cardíaca fora dos limites especificados



Fonte: Elaborado pelo autor

4.4 Monitoramento de frequência por GATT

O monitoramento da frequência cardíaca foi baseado no projeto *Android BLE Connect*, aplicativo de licença permissiva, definida pela licença Apache 2.0(*APACHE LICENSE*, 2004), onde o mesmo era requisitado em um aparelho da marca *Xiaomi*, *Mi Band* 2, e é registrado em um smartphone *Android* com conexão, via *bluetooth low energy*, com o aparelho já pareado.

Baseado nisso e nos testes já verificados no aparelho genérico B57, se torna uma possibilidade que o sistema consiga capturar qualquer sistema que se comporte com o protocolo *GATT* e que tenha sensor com seus serviços e características, capaz de ler medidas de frequência cardíaca, podendo assim se tornar um estudo futuro em outros tipos de aparelho, como frequencímetros anatomicamente feitos para outras partes do corpo, como tronco, mãos e pernas, afim de expandir tais medições e melhorar os tipos de monitoramento.

O sistema verifica se o nome do aparelho digitado, está conectado e pareado ao dispositivo mobile. Em caso positivo, é feita a conexão por meio do *BluetoothService,* onde, ao iniciar o serviço, é feito o pareamento do dispositivo pelo seu endereço *MAC* e com isso feita a

conexão com *callback* do *GATT* por meio da função *connectGatt,* que permite esse acoplamento por meio do endereço *MAC*, derivada da *BluetoothDevice*, iniciado no construtor do arquivo *BLE*, sendo este uma classe objeto.

Em caso de positivo para o dispositivo pareado, o serviço inicia o processo de verificação da medida da frequência cardíaca, utilizando do sistema de identificadores únicos universais, ou *UUID*, implementado pela Bluetooth.

No projeto, são armazenados os *UUIDs* necessários para colher a amostra gerada pelo smartwatch, como esses *UUIDs* são específicos para cada tipo de serviço, o serviço de frequência cardíaca, tem seu próprio identificador, sendo ele: 0x18d, mantendo o sufixo de identificadores de 16 *bits* inerentes da *bluetooth*.

A partir disso, o serviço inicia o processo para coletar os dados da característica do serviço implementado para a frequência cardíaca, mantendo o intuito em caso de mudanças nas características mensuradas (por meio das funções "*onCharacteristicRead*" e "*onCharacteristicChanged*"), e envia os dados por meio do broadcast, enviando assim, o valor da frequência cardíaca medida para o aplicativo e assim, para o banco de dados.

4.5 Objeto *BLE*

O objeto BLE mantém a inicializações das classes necessárias para utilização do sistema de utilização do *bluetooth low energy* utilizados no projeto.

4.5.1 *BluetoothDevice*

Representando um dispositivo remoto de *Bluetooth*, a classe *BluetoothDevice,* permite a criação de uma conexão entre o dispositivo respectivo ou fazer consultas sobre suas informações, entre elas, o nome, endereço e estado de conexão, sendo os objetos dessa classe imutáveis e suas operações feitas por meio do *BluetoothAdapter*, utilizado para a criação do devido *BluetoothDevice* (*ANDROID DEVELOPERS*, 2022).

4.5.2 *BluetoothAdapter*

Representa o adaptador *Bluetooth* do dispositivo local, permitindo tarefas fundamentais de *Bluetooth,* como, iniciar a procura por dispositivos *bluetooth*, listar os dispositivos pareados,

instanciar a classe *BluetoothDevice,* usando um endereço *MAC* conhecido, entre outras tarefas (*ANDROID DEVELOPERS*, 2022).

4.5.3 *BluetoothGatt*

API pública para adequação de perfil *GATT*, fornecendo meio de comunicação com dispositivos funcionais *bluetooth low energy* (*ANDROID DEVELOPERS*, 2022).

4.5.4 *WearableAddress*

Este valor, se dá pelo endereço *MAC* guardado, em uma variável do tipo *string*, também, o endereço encontrado do dispositivo ao qual foi procurado, tornando possível passá-lo do procedimento do *BluetoothFragment*, para o *BluetoothService*, por meio dessa classe objeto.

4.5.5 Nome do usuário

Este valor se inicia como nulo, e a partir do momento que há um monitoramento de um paciente no banco de dados ele é subscrito pelo nome da pasta do paciente que no caso consiste em no nome completo do paciente monitorado.

4.6 Funcionamento do *APP*

O funcionamento do APP é proposto de duas formas: uma é gerenciando o processo e alteração dos pacientes, fazendo suas mensurações e outra é o monitoramento. O propósito do projeto é que haja um cuidador (ou o próprio paciente) e um cuidador da área da saúde. O cuidador, faz as medições dos pacientes em seu âmbito, utilizando o aparelho smartwatch para tal. Esses dados são enviados para a nuvem e o atuante da área da saúde, analisa tais dados, pelo monitoramento, dentro do mesmo aplicativo e no mesmo âmbito de *login* do cuidador e verifica se há algo que possa ser tratado ou que deverá ter mais atenção em relação a seus pacientes, como visto na figura 27.

Figura 27: Fluxograma de funcionamento do *APP*

Fonte: Elaborado pelo autor.

O APP não substitui os meios de tratamento de saúde e não proporciona o mesmo, já que é necessário um atuante da área da saúde para que haja sucesso no seu propósito de criação.

Para o funcionamento dos alertas, o app foi testado com limites da sessão de monitoramento com êxito em evidenciar se a frequências medida no momento estava dentro das fronteiras de média aplicadas.

5 CONCLUSÃO

Neste capítulo serão evidenciadas as experiências obtidas ao fim da pesquisa de dados para o início do projeto final, os aprendizados e as dificuldades obtidas no decorrer.

5.1 Considerações finais

O presente trabalho é a implementação de um sistema de monitoramento para melhor ajudar pessoas com o auxílio de terceiros, para que tenham uma melhor chance e forma de socorro, obtendo um resultado satisfatório, pelo fato de que foi possível a verificação em tempo real com notificação, caso se tenha uma alta ou uma queda nos sinais vitais (batimentos cardíacos) do usuário que se está monitorando.

Com o passar do processo de desenvolvimento e pesquisa, foram analisados várias *APIs* com sistemas de *GATT* para base do monitoramento de frequência cardíaca, sendo a base principal deste projeto, o sistema *Android BLE Connect.* Porém, foram vistos inúmeros tipos de projetos, que utilizavam do *GATT* em diversos tipos de sistemas e outros sensores, evidenciando sua capacidade e longevidade, passando por vários tipos de tecnologias e sendo utilizada até os dias atuais com um peso enorme no mercado digital, com maior número de aplicações disponíveis para pesquisa datados e muitas vezes depreciados.

Também foram observados os inúmeros sistemas de obtenção de resultados de melhorias de saúde, para serem utilizados em dispositivos com intuito total de exercícios físicos e nutricionais, tendo sistemas de por esportes, mas também de calorias consumidas, que é uma tendência grande para os tempos atuais.

Foram cogitados também, sistemas de monitoramento cardíaco no sistema operacional *smartwatch WearOs,* afim de maior disponibilidades de para melhorias aliadas as modernidades de um sistema mais novo, porém para esse tipo de análise foi realizado por meio dos emuladores disponibilizados pelo *AndroidStudio,* sem qualquer tipo de feedback dos sensores, já que não era possível fazer tais medições, escolhendo assim o dispositivo genérico, que traz maior possibilidade de abrangência ao público, dado seu valor de mercado mais baixo.

Com o passar da pesquisa foi-se observando, também, o poderio da linguagem de programação Java e Kotlin, mais utilizadas na plataforma do *AndroidStudio.* Além de serem linguagens com uma abstração elevada, o que permite maiores flexibilidades, mantém também uma curva de aprendizado menos complexa, com os tutoriais explicativos do site de desenvolvedores da *Google.*

Durante esse processo de desenvolvimentos, também foi visto o enorme valor da ferramenta da plataforma do *AndroidStudio*, que tornou possível a implementação de várias *APIs*, que facilitaram e facilitam o processo de programação de aplicativos e jogos para a comunidade Android, visto como uma delas a plataforma do *Firebase.*

A plataforma *Firebase* foi de grande importância para implementar o monitoramento remoto, já que seria necessária uma implementação de banco de dados, na qual ela se mostrou uma excelente fonte de funções, às quais foram fortemente utilizadas e aproveitadas neste trabalho. Suas regras de acesso no console são de fácil entendimento, e suas funções implementadas dentro do aplicativo são facilmente aprendidas e utilizadas, sem grande esforço de acesso às coleções e documento. Sua forma de amostragem de dados em tempo real foi essencial para a implementação do trabalho aqui descrito, mantendo os cuidados monitorando o paciente, com muita precisão e sem *delays* ou atrasos.

O objetivo deste trabalho é alcançar um estudo para que se pudesse desenvolver o conhecimento necessário, obtendo as informações chaves para que o projeto pudesse ser realizado com completude.

O presente trabalho criar uma ferramenta capaz de prestar auxílio a pessoas que tenham dificuldades de assistência plena e constante com problemas relacionados com a oxigenação e pressão arterial, a fim de criar um mecanismo de emergência de forma a avisar o médico ou cuidador permitindo um amparo mais efetivo para melhor qualidade de vida, com menos clausura.

O desenvolvimento do presente trabalho possibilitou uma análise aprofundada nas inúmeras possibilidades de engajamento do *smartwatch* na vida em artigos e temas relacionados ao procedimento esperado no deste (descritos no capítulo de trabalhos relacionados). Foram verificados também, vários tipos de dispositivos com mais módulos implementados nos mercados *online* que não têm tanta dependência do aparelho celular, porém com alto custo sem fácil acessibilidade, por isso, com relação ao objetivo deste trabalho foram escolhidos módulos que se encontram na maioria dos dispositivos encontrados no mercado e de baixo custo, para maior expansão de pesquisa, pós desenvolvimento do *software* e em para trabalhos futuros haver maior base de dados.

Entre os maiores desafios no decorrer do projeto, está a informação de hardware do *smartwatch* e de seus sensores. Como é um projeto de várias indústrias e marcas, sua elaboração não é muito difundida para o público. Foram encontrados projetos com hardware aberto na *internet* e artigo ao qual o hardware é construído (MOREIRA; LEAL, 2019), porém tendo como principal problema, o alto consumo de energia e monetário, pois ficariam mais caros que um *smartwatch* simples com mais sensores.

Este problema foi contornado pelo projeto que faz a conexão *GATT* pelo endereço *MAC* e pelo nome do dispositivo já conectado ao sistema, permitindo uma ligação mais simples, já aguardando a coleta das características pelo *UUID* da frequência cardíaca e podendo administrar esse valor coletado.

O projeto que aqui segue, possibilitou um enorme conhecimento nos sistemas mobile e de smartwatch, mais aprofundado no *WearOs*, tanto no campo de desenvolvimento do *APP*, quanto na sua estruturação de ciclo de vida, de atividades e fragmentos, que são inúmeras vezes visíveis em outros aplicativos do dia-a-dia.

Com a capacidade do *GATT*, também é possível verificar tipos de exercícios, quedas e outros tipos de medidas vitais, que tornam o monitoramento mais preciso e mais efetivo em situações emergenciais, podendo prover maiores cuidados com o nicho de pacientes que se aplica o monitoramento. Como o protocolo *GATT* não necessita de um SO específico, o projeto pode evoluir ainda para outros tipos de smartphones e até sistema computacionais em *WEB*, potencializando sua redundância e tornando um sistema mais robusto e flexível com os locais onde poderá ser implementado.

Conclui-se, que a utilização dos smartwatches já vem sendo amplamente explorada, com um grande potencial para que haja maior suporte nos quesitos que são possíveis estimar com seu *hardware* rico em sensores.

5.2 Trabalhos futuros

Como possíveis trabalhos futuros, sugere-se:

- Fazer medições de voluntários para obter base de dados de pesquisa.
- Fazer a implementação de segurança em relação aos dados do usuário criando um banco de dados;
- Fazer implementação de mais tipos de sensores, como acelerômetro, giroscópio, oxímetro, termômetro e sensor de pressão arterial, por meio do *GATT;*
- Implementar os métodos de medições integradas à um *Wearos* (Sistema operacional de *smartwatches Google*), para um maior controle longínquo dos momentos de medições, sem a necessidade de o usuário acionar o dispositivo;
- Implementar sistema de *GPS* ao aplicativo, para abranger um campo maior de monitoramento;
- Utilizar métodos de criptografia para maior segurança dos dados dos pacientes e cuidadores.;

- Implementar novas formas de login e cadastro ao aplicativo, ampliando o número de utilizadores do sistema;
- Analisar dados de pessoas utilizando os sensores de smartwatches e fazer um banco de dados para melhor analise da frequência cardíaca;
- Utilizar outros tipos de sistemas vestíveis para analise de dados e verificar qual seria o mais confortável ou de maior valia para o sistema;
- Implementar o projeto em sistemas Web para maior abrangência de sistemas e nichos.
- Implementar avisos sonoros e ligação de emergência para atendimento a urgências de pacientes

Nos apêndices se encontram os códigos do programa, e em Anexo 1 a autorização de publicação o trabalho de conclusão de curso.

REFERÊNCIAS

ANDROID STUDIO. **Conheça o Android Studio**. 2021. Disponível em: <https://developer.android.com/studio/intro?hl=pt-br>. Acesso em: 23 abril. 2022.

APACHE LICENSE. Apache license version 2.0. 2004. Disponível em: <https://www.apache.org/licenses/LICENSE-2.0>. Acesso em: 23 abril. 2022.

AMAZON. 2022. Disponível em: <https://www.amazon.com.br/Relógio-Smartwatch-Hero-Band-Android/dp/B0851WSJ7D> Acesso em 16 de novembro de 2022.

ARAUJO, Regina Borges. Computação Ubíqua, Princípios, Tecnologias e Desafios – XXI Simpósio Brasileiro de Redes de Computadores. 2003. <https://www.academia.edu/2167004/Computação_ubíqua_Princípios_tecnologias_e_desafios>. Acesso em: 14 maio 2022.

BEZERRA, C. C.; RODRIGUES FILHO, D. C.; GALEANO, N.; EBERHARDT, P. L.; JARÁ, S. N.; RIBEIRO, G. L. S. **Avaliação do desmatamento em área de preservação permanente e reconhecimento dos locais de sumidouro e ressurgência em um trecho do córrego facão em Cáceres, MT**. In: SIMPÓSIO DE GEOTECNOLOGIAS NO PANTANAL, 1. (GEOPANTANAL)., 2006, Campo Grande. Anais**...** Campinas: Embrapa Informática Agropecuária; São José dos Campos: INPE, 2006. p. 424-430. CD-ROM. ISBN 85-17-00029-3. Disponível em: <http://urlib.net/ibi/6qtX3pFwXQZGivnK2Y/NqGEe>. Acesso em: 25 abr. 2022

BLUETOOTH Wireless Technology. ***Bluetooth***, c2022. Disponível em: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/. Acesso em: 13 maio 2022.

COSTA, K. N. T.; DOS SANTOS, C. T.; SANTOS, F. A. N. V. dos. **Smartphone X smartwatch: uma análise Comparativa de Aspectos Ergonômicos de Usabilidade de interfaces**. **Anais do Congresso Internacional de Conhecimento e Inovação – ciki**, *[S. l.]*, v. 1, n. 1, 2019. Disponível em: <https://proceeding.ciki.ufsc.br/index.php/ciki/article/view/690>. Acesso em: 3 maio. 2022.

ELMASRI, Ramez; NAVATHE, Shamkant B. Sistemas de Banco de Dados. 4. Ed. São Paulo: Pearson Education, 2005.

DAVENPORT, Tom. ***The Analytics of Things***, 2015. Disponível em: < https://www.linkedin.com/pulse/analytics-things-tom-davenport/>. Acesso em: 13 maio 2022.

DAVIDSON, Robert *et al*. GATT (Services and Characteristics). *In*: DAVIDSON, Robert *et al*. ***Getting Started with Bluetooth Low Energy***. [*S. l.*]: O'Reilly Media, Inc., 2014. cap. 4. Disponível em: <https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch04.html>. Acesso em: 13 maio 2022.

FIREBASE Realtime Database. **Firebase**, 2021. Disponível em: <https://firebase.google.com/docs/database?hl=pt-br>. Acesso em: 13 maio 2022.

FIREBASE Realtime Database. **Firebase**, 2021. Disponível em: <https://firebase.google.com/products/firestore>. Acesso em: 16 novembro 2022.

HARVEST NoSQL Speed With The Combination Of PHP, **Clarion Technologies**, 2020. Disponível em: <https://www.clariontech.com/blog/harvest-nosql-speed-with-the-combination-of-php>. Acesso em: 13 maio. 2022.

KIM, Seulgi. **Android Sample Application to connect BLE device.** 2020. Disponível em <https://github.com/4z7l/Android-BLE-Connect.git>. Acesso em 16 de novembro de 2022.

MAGRANI, Eduardo. **A Internet das coisas**. Rio de Janeiro: Fgv Editora, p. 46, 2018. 192 f. Disponível em: <http://bibliotecadigital.fgv.br/dspace/bitstream/handle/10438/23898/A%20internet%20das%20coisas.pdf?sequence=1&isAllowed=y>. Acesso em: 23 abr. 2022.

MOREIRA, Mathias Custódio; LEAL, Daniel Martins. **Desenvolvimento de um smartwatch para pessoas com deficiência visual**. 2019. 45 f. Trabalho de Conclusão de Curso (Graduação) - Escola de Engenharia Elétrica, Mecânica e da Computação, Universidade Federal de Goiás, Goiânia, 2019.

O que é IoT?. **Oracle**, c2022. Disponível em https://www.oracle.com/br/internet-of-things/what-is-iot/#business-decisions-iot. Acesso em 13 maio 2022.

Paschoal, MA et al. **Variabilidade da freqüência cardíaca em diferentes faixas etárias.** Brazilian Journal of Physical Therapy [online]. 2006, v. 10, n. 4 [Acessado 25 Outubro 2022] , pp. 413-419. Disponível em: <https://doi.org/10.1590/S1413-35552006000400009>. Epub 16 Jan 2007. ISSN 1809-9246. https://doi.org/10.1590/S1413-35552006000400009.

PEREIRA, Carlos. **VARIABILIDADE DA FREQUÊNCIA CARDÍACA E SISTEMA NERVOSO AUTÔNOMO EM RATOS**: UM ESTUDO COM CORAÇÃO ISOLADO E MANIPULAÇÃO FARMACOLÓGICA IN VIVO. 2018. Trabalho de Mestrado (Mestrado em Filosofia) - Universidade Federal do Paraná, Curitiba, 2018. Disponível em: https://acervodigital.ufpr.br/bitstream/handle/1884/56555/R%20-%20D%20-%20CARLOS%20HENRIQUE%20PEREIRA.pdf?sequence=1&isAllowed=y. Acesso em: 14 jun. 2022.

PÉREZ, Francisco de Arriba; RODRÍGUEZ, Manuel Caeiro; GAGO, Juan M. Santos. ***Collection and Processing of Data from Wrist Wearable Devices in Heterogeneous and Multiple-User Scenarios***. *Sensors*, Vigo, p. 1-31, 21 set. 2016. DOI 10.3390/s16091538. Disponível em: <https://www.mdpi.com/1424-8220/16/9/1538> Acesso em: 7 maio 2022.

SANTOS, Pedro Miguel Pereira. **Internet das coisas: o desafio da privacidade.** Dissertação (mestrado em sistemas de informação organizacionais) - Escola Superior de Ciências Empresariais, Instituto Politécnico de Setúbal, 2016. Disponível em: <https://comum.rcaap.pt/bitstream/10400.26/17545/1/Disserta%C3%A7%C3%A3o%20Pedro%20Santos%20140313004%20MSIO.pdf>. Acesso em: 23 abr. 2022.

SDK | *Kit de Desenvolvimento de Software*. ***Red Hat,*** 2020. Disponível em: <https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-SDK#:~:text=SDK%20é%20a%20sigla%20para,operacional%20ou%20linguagem%20de%20programação>. Acesso em: 13 maio 2022

SILVA, Leoni; FERNANDES, Sérgio; LINS, Robson. **Uso de SmartWatch no Auxílio a Monitoração de Arritmias Cardíacas.** Brazilian Journal of Development, Curitiba, v. 6, n. 10, p. 75511-75525, 6 out. 2020. DOI 10.34117/bjdv6n10-108. Disponível em:

<https://www.brazilianjournals.com/index.php/BRJD/article/view/17844/14454>. Acesso em: 26 abr. 2022.

SOBRAC – Sociedade Brasileira de Arritmias Cardíacas. **Release dia mundial do coração e a necessidade de atenção paras as arritmias cardíacas.** Disponível em: <https://sobrac.org/home/release-dia-mundial-do-coracao-e-a-necessidade-de-atencao-paras-as-arritmias-cardiacas> Acessado em: 30 de abr. de 2022.

SPACE Segment. ***GPS,*** 2021. Disponível em: < https://www.gps.gov/systems/gps/space/>. Acesso em: 13 maio 2022.

TAVARES, Victor. ***SafeWatch***: Detectando quedas com Smartwatches. Orientador: Vaninha Vieira dos Santos. 2016. 77 p. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Federal da Bahia, Salvador, 2016. Disponível em: <https://repositorio.ufba.br/bitstream/ri/20949/1/mono_final.pdf>. Acesso em: 26 abr. 2022.

TOWNSEND, Kevin. ***Introduction to Bluetooth Low Energy****: A basic overview of key concepts for* BLE. [*S. l.*], 20 mar. 2014. Disponível em: <https://learn.adafruit.com/introduction-to-bluetooth-low-energy/introduction>. Acesso em: 26 abr. 2022.

VALER, Rafael. **Estudo de interação e implementação de um aplicativo para smartwatch**. 2017. Monografia (Bacharelado em Ciência da Computação) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 2017. Disponível em: <https://lume.ufrgs.br/handle/10183/175069>. Acesso em: 14 maio 2022.

6 APÊNDICES

Apêndice A

Implementação em Kotlin da Principal Activity

PrincipalActivity.kt

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.fragment.app.Fragment
import com.example.tcc2.R
import com.example.tcc2.databinding.ActivityPrincipalBinding

class Principal : AppCompatActivity() {

    private lateinit var binding: ActivityPrincipalBinding

    private val optionFragmento = Option()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityPrincipalBinding.inflate(layoutInflater)

        trocarFragmento(optionFragmento)

        setContentView(binding.root)
    }

    private fun trocarFragmento(fragmento: Fragment) {
        supportFragmentManager.beginTransaction()
            .replace(R.id.Prin_frame, fragmento)
            .commit()
    }

}
```

Apêndice B

Implementação em Kotlin do fragmento Bluetooth

BluetoothFragment.kt

```
import android.app.ProgressDialog
import android.bluetooth.BluetoothDevice
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.content.IntentFilter
import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.fragment.app.Fragment
import com.example.tcc2.databinding.FragmentBluetoothBinding
import com.example.tcc2.form.tcc.util.*
import com.google.firebase.firestore.FirebaseFirestore

// Alterado por Gabriel de Paiva em 11/08/22


class BluetoothFragment : Fragment() {

    private lateinit var binding: FragmentBluetoothBinding
    private lateinit var progressDialog: ProgressDialog
    private lateinit var broadcastReceiver: BroadcastReceiver
    private val db = FirebaseFirestore.getInstance().collection(BLE.authUid.toString())
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {

        binding = FragmentBluetoothBinding.inflate(inflater, container, false)

        binding.MedBtnVerificarPac.setOnClickListener {
            if (!BLE.bluetoothAdapter.isEnabled) {
                Toast.makeText(context, "Conecte-se ao dispositivo de medição",
Toast.LENGTH_LONG)
                    .show()
            } else {

                val nomePac = binding.MedTxtPaciente.text.toString()
                if (nomePac.isEmpty()) {
                    Toast.makeText(
                        context,
                        "Coloque o nome do paciente, para fazer medição!",
                        Toast.LENGTH_SHORT
```

```
            ).show()
        } else {
            db.document(nomePac).get()
                .addOnCompleteListener() {
                    if (it.getResult().exists()) {
                        Toast.makeText(
                            context,
                            "Paciente encontrado com sucesso!",
                            Toast.LENGTH_SHORT
                        ).show()
                        BLE.nomeBD = nomePac
                        db.document(nomePac).addSnapshotListener { snapshot, e ->
                            if (e != null) {
                                Toast.makeText(context, "ERRO", Toast.LENGTH_SHORT).show()
                                return@addSnapshotListener
                            }
                            if (snapshot != null && snapshot.exists()) {
                                BLE.media = snapshot.get("media").toString().toFloat()
                            }
                        }
                    } else {
                        Toast.makeText(
                            context,
                            "Paciente não existe nesse âmbito",
                            Toast.LENGTH_SHORT
                        ).show()
                    }
                }
        }
    }
}

binding.MedBtnVerificar.setOnClickListener {
    val devices: Set<BluetoothDevice> = BLE.bluetoothAdapter.bondedDevices
    val nome = binding.MedTxtNome.text.toString()
    if (nome.isEmpty()) {
        Toast.makeText(
            context,
            "Coloque o nome do dispositivo pareado",
            Toast.LENGTH_SHORT
        )
            .show()
    } else {
        for (device in devices) {
            if (device.name.contains(nome)) {
                Log.i("Conexão", "Foi encontrado")
                device.address.also {
                    BLE.deviceAddress = it
```

```
                }
            } else {
                Log.i("Conexão", "Nao foi encontrado")
            }
        }
        if (BLE.deviceAddress != null) {
            binding.MedTxtDeviceName.text = BLE.deviceAddress
            Toast.makeText(
                context,
                "Dispositivo $nome encontrado e pronto para fazer medição",
                Toast.LENGTH_SHORT
            ).show()
            initBLEObject()
            initProgressDialog()
            initView()
        } else {
            Toast.makeText(
                context,
                "Dispositivo $nome não encontrado",
                Toast.LENGTH_SHORT
            )
                .show()
        }
    }

    return binding.root
}

override fun onStart() {
    super.onStart()
    registerReceiver()
}

override fun onStop() {
    super.onStop()
    requireActivity().run {
        unregisterReceiver(broadcastReceiver)
        stopService(Intent(this, BluetoothService::class.java))

    }
}

private fun registerReceiver() {
    broadcastReceiver = object : BroadcastReceiver() {
        override fun onReceive(context: Context?, intent: Intent) {
            when (intent.action) {
                ACTION_GATT_CONNECTED -> stateConnected()
```

```
                ACTION_GATT_DISCONNECTED -> stateDisConnected()
                ACTION_DATA_AVAILABLE -> displayData(intent)

            }
        }
    }
    val filter = IntentFilter()
    filter.apply {
        addAction(ACTION_GATT_CONNECTED)
        addAction(ACTION_GATT_DISCONNECTED)
        addAction(ACTION_DATA_AVAILABLE)
    }
    requireActivity().registerReceiver(broadcastReceiver, filter)
}

private fun initView() {
    if (!isServiceStarted(requireContext()))
        stateDisConnected()

}

private fun initBLEObject() {
    requireActivity().startService(
        Intent(
            requireContext(),
            BluetoothService::class.java
        )
    )
}

private fun initProgressDialog() {
    progressDialog = ProgressDialog(requireContext())
    progressDialog.apply {
        setTitle("Bluetooth Low Energy")
        setMessage("Connecting...")
        setCancelable(true)
        create()
    }
}

private fun stateConnected() {
    progressDialog.dismiss()
    setServiceState(requireContext(), true)
}

private fun stateDisConnected() {
    progressDialog.show()
    setServiceState(requireContext(), false)
```

```
    }

    private fun displayData(intent: Intent) {
        val heartRate = intent.getIntExtra(HEART_RATE, 0)
        BLE.bpm = heartRate
        var media: Float?
        media = BLE.media
        if (media != null) {
            binding.MedTvMedia.text = media.toInt().toString()
        }
        if (binding.switchMedia.isChecked) {
            media = (BLE.media?.plus(heartRate))?.div(2)
        }
        binding.MedTvBatimento.text = heartRate.toString()
        db.document(BLE.nomeBD.toString())
            .update(
                mapOf(
                    "bpm" to heartRate,
                    "media" to media
                )
            )

    }
}
```

Apêndice C

Implementação em Kotlin do fragmento Option

Option.kt

```
import android.content.Intent
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.example.tcc2.R
import com.example.tcc2.databinding.FragmentOptionBinding
import com.example.tcc2.form.tcc.login.Login
import com.google.firebase.auth.FirebaseAuth

class Option : Fragment() {
    private val gerenciarFragmento = GerenciarFragment()
    private val monitorarFragmento = TelaMonitoramento()
```

```
    private val medirFragmento = BluetoothFragment()

    private lateinit var binding: FragmentOptionBinding
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {


        binding = FragmentOptionBinding.inflate(inflater, container, false)

        binding.gerenciarBtn.setOnClickListener {
            trocarFragmento(gerenciarFragmento)

        }
        binding.medirBtn.setOnClickListener {
            trocarFragmento(medirFragmento)
        }

        binding.monitorBtn.setOnClickListener {
            trocarFragmento(monitorarFragmento)
        }

        binding.sairBtn.setOnClickListener {
            irTelaLogin()
        }
        return binding.root
    }

    private fun trocarFragmento(fragmento: Fragment) {
        requireActivity().supportFragmentManager.beginTransaction()
            .replace(R.id.Prin_frame, fragmento)
            .addToBackStack("ultimo")
            .commit()
    }

    private fun irTelaLogin() {
        FirebaseAuth.getInstance().signOut()
        val intent = Intent(context, Login::class.java)
        startActivity(intent)
    }

}
```

Apêndice D

Implementação em Kotlin do fragmento Gerenciar

GerenciarFragment.kt

```
package com.example.tcc2.form.tcc.principal

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.example.tcc2.R
import com.example.tcc2.databinding.FragmentGerenciarBinding

class GerenciarFragment : Fragment() {
    private lateinit var binding: FragmentGerenciarBinding
    private val pacienteAdFragmento = TelaPaciente()
    private val pacienteAtFragmento = TelaPacienteAtt()
    private val pacienteExFragmento = TelaPacienteEx()

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentGerenciarBinding.inflate(inflater, container, false)

        binding.GrBtnAdicionar.setOnClickListener {
            trocarFragmento(pacienteAdFragmento)
        }
        binding.GrBtnAtualizar.setOnClickListener {
            trocarFragmento(pacienteAtFragmento)
        }
        binding.GrBtnExcluir.setOnClickListener {
            trocarFragmento(pacienteExFragmento)
        }

        return binding.root
    }

    private fun trocarFragmento(fragmento: Fragment) {
        requireActivity().supportFragmentManager.beginTransaction()
            .replace(R.id.Prin_frame, fragmento)
            .addToBackStack("ultimo")
            .commit()
    }
}
```

Apêndice E

Implementação em Kotlin do fragmento TelaPaciente

TelaPaciente.kt

```
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import com.example.tcc2.databinding.FragmentTelaPacienteBinding
import com.example.tcc2.form.tcc.util.BLE
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase


class TelaPaciente : Fragment() {
    private val colecaoPaciente = Firebase.firestore.collection(BLE.authUid.toString())
    private lateinit var binding: FragmentTelaPacienteBinding

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentTelaPacienteBinding.inflate(inflater, container, false)
        binding.btnCadastro.setOnClickListener {

            val idade = binding.txtfireidade.text.toString()
            val media = binding.txtfiremedia.text.toString()
            val bpm = binding.txtfirebpm.text.toString()
            val nome = binding.txtfirenome.text.toString()


            if (campoVazio(nome) || campoVazio(media) || campoVazio(bpm) || campoVazio(
                    idade
                )
            ) {
                Toast.makeText(
                    context,
                    "Considere preencher todos os campos",
                    Toast.LENGTH_SHORT
                )
```

```
            .show()
        } else {
            val builder = AlertDialog.Builder(requireContext())
            builder.setTitle("Salvar Paciente")
            builder.setMessage("Tem certeza que deseja salvar esses dados")
            builder.setPositiveButton("Sim") { dialog, which ->
                val usuario = Usuarios(nome, idade.toInt(), media.toFloat(), bpm.toInt())
                salvarPaciente(usuario)
            }
            builder.setNegativeButton("Não") { dialog, which ->
                dialog.dismiss()
            }
            val dialog: AlertDialog = builder.create()
            dialog.show()
        }
    }

    return binding.root
}

private fun salvarPaciente(paciente: Usuarios) {
    colecaoPaciente.document(paciente.nome).get()
        .addOnCompleteListener() {
            if (it.getResult().exists()) {
                Toast.makeText(
                    context,
                    "O documento já existe neste login, utilize as abas de alteração ou exclusão",
                    Toast.LENGTH_LONG
                ).show()
            } else {
                colecaoPaciente.document(paciente.nome)
                    .set(paciente)
                    .addOnSuccessListener {
                        Toast.makeText(activity, "Salvo com sucesso!", Toast.LENGTH_LONG)
                            .show()
                    }
                    .addOnFailureListener {
                        Toast.makeText(activity, "Falha ao Salvar!", Toast.LENGTH_LONG)
                            .show()
                    }
            }
        }
}

private fun campoVazio(valor: String): Boolean {
    return valor == "" || valor.isEmpty() || valor.isBlank()
}
```

```
}
```

Apêndice F

Implementação em Kotlin do fragmento TelaPacienteAtt

TelaPacienteAtt.kt

```
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.View.INVISIBLE
import android.view.View.VISIBLE
import android.view.ViewGroup
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import com.example.tcc2.databinding.FragmentTelaPacienteAttBinding
import com.example.tcc2.form.tcc.util.BLE
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase

class TelaPacienteAtt : Fragment() {
    private val colecaoPaciente = Firebase.firestore.collection(BLE.authUid.toString())
    private lateinit var binding: FragmentTelaPacienteAttBinding
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentTelaPacienteAttBinding.inflate(inflater, container, false)
        binding.AttBtnAtualizar.visibility = INVISIBLE
        binding.AttBtnVerificar.setOnClickListener {
            val nome = binding.AttEdtNome.text.toString()
            BLE.nomeBD = nome
            if (campoVazio(nome)) {
                Toast.makeText(
                    context,
                    "Campo de nome vazio, digite um nome válido",
                    Toast.LENGTH_SHORT
                )
                    .show()
            } else {
                colecaoPaciente.document(nome).get()
                    .addOnCompleteListener() {
```

```
            if (it.getResult().exists()) {
                binding.AttBtnAtualizar.visibility = VISIBLE
                colecaoPaciente.document(nome).addSnapshotListener { snapshot, e ->
                    if (e != null) {
                        Toast.makeText(context, "ERRO", Toast.LENGTH_SHORT).show()
                        return@addSnapshotListener
                    }
                    if (snapshot != null && snapshot.exists()) {

                        val nome = snapshot.get("nome").toString()
                        val idade = snapshot.get("idade").toString()
                        val bpm = snapshot.get("bpm").toString()
                        val media = snapshot.get("media").toString()

                        binding.AttTvNome.setText(nome)
                        binding.AttTvIdade.setText(idade)
                        binding.AttTvBpm.setText(bpm)
                        binding.AttTvMedia.setText(media)
                    }
                }
            } else {
                Toast.makeText(
                    context,
                    "Paciente não existente nesse âmbito, tente outro nome",
                    Toast.LENGTH_SHORT
                )
                    .show()
            }
        }
    }
}
binding.AttBtnAtualizar.setOnClickListener {
    val nome = BLE.nomeBD
    val builder = AlertDialog.Builder(requireContext())
    builder.setTitle("Atualização de Paciente: $nome")
    builder.setMessage("Tem certeza que deseja atualizar os dados do paciente?")
    builder.setPositiveButton("Sim") { dialog, which ->
        val nome = binding.AttTvNome.text.toString()
        val idade = binding.AttTvIdade.text.toString().toInt()
        val bpm = binding.AttTvBpm.text.toString().toInt()
        val media = binding.AttTvMedia.text.toString().toFloat()

        val updates = hashMapOf<String, Any>(
            "nome" to nome,
            "idade" to idade,
            "bpm" to bpm,
            "media" to media
        )
```

```
            colecaoPaciente.document(BLE.nomeBD.toString()).update(updates)
                .addOnSuccessListener {
                    binding.AttBtnAtualizar.visibility = INVISIBLE

                    binding.AttEdtNome.setText("")

                    binding.AttTvNome.setText("")
                    binding.AttTvBpm.setText("")
                    binding.AttTvIdade.setText("")
                    binding.AttTvMedia.setText("")

                    getActivity()?.getSupportFragmentManager()!!.popBackStack();
                    Toast.makeText(
                        context,
                        "Campos atualizados com sucesso",
                        Toast.LENGTH_SHORT
                    ).show()

                }
                .addOnFailureListener { e ->
                    Toast.makeText(
                        context,
                        "Falha na operação",
                        Toast.LENGTH_SHORT
                    ).show()
                }


        }
        builder.setNegativeButton("Não") { dialog, which ->
            dialog.dismiss()
            Toast.makeText(
                requireContext(), "Atualização cancelada.",
                Toast.LENGTH_SHORT
            ).show()
        }
        val dialog: AlertDialog = builder.create()
        dialog.show()

    }

    return binding.root
}

private fun campoVazio(valor: String): Boolean {
    return valor == "" || valor.isEmpty() || valor.isBlank()
}
```

```
}
```

Apêndice G

Implementação em Kotlin do fragmento TelaPacienteEx

TelaPacienteEx.kt

```
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.View.INVISIBLE
import android.view.View.VISIBLE
import android.view.ViewGroup
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import com.example.tcc2.databinding.FragmentTelaPacienteExBinding
import com.example.tcc2.form.tcc.util.BLE
import com.google.firebase.firestore.FieldValue
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase

class TelaPacienteEx : Fragment() {
    private val colecaoPaciente = Firebase.firestore.collection(BLE.authUid.toString())
    lateinit var binding: FragmentTelaPacienteExBinding
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentTelaPacienteExBinding.inflate(inflater, container, false)
        binding.ExBtnExcluir.visibility=INVISIBLE
        binding.ExBtnVerificar.setOnClickListener {
            val nome = binding.ExEdtNome.text.toString()
            BLE.nomeBD = nome
            if (campoVazio(nome)) {
                Toast.makeText(
                    context,
                    "Campo de nome vazio, digite um nome válido",
                    Toast.LENGTH_SHORT
                )
                    .show()
            } else {
                colecaoPaciente.document(nome).get()
                    .addOnCompleteListener() {
```

```
                if (it.getResult().exists()) {
                    binding.ExBtnExcluir.visibility=VISIBLE
                    colecaoPaciente.document(nome).addSnapshotListener { snapshot, e ->
                        if (e != null) {
                            Toast.makeText(context, "ERRO", Toast.LENGTH_SHORT).show()
                            return@addSnapshotListener
                        }
                        if (snapshot != null && snapshot.exists()) {

                            val nome = snapshot.get("nome").toString()
                            val idade = snapshot.get("idade").toString()
                            val bpm = snapshot.get("bpm").toString()
                            val media = snapshot.get("media").toString()

                            binding.ExTvNome.text = nome
                            binding.ExTvIdade.text = idade
                            binding.ExTvBpm.text = bpm
                            binding.ExTvMedia.text = media
                        }
                    }
                } else {
                    Toast.makeText(
                        context,
                        "Paciente não existente nesse âmbito, tente outro nome",
                        Toast.LENGTH_SHORT
                    )
                        .show()
                }
            }
            binding.ExBtnExcluir.setOnClickListener {
                val builder = AlertDialog.Builder(requireContext())
                builder.setTitle("Exclusão de Paciente: $nome")
                builder.setMessage("Tem certeza que deseja excluir os dados do paciente?")
                builder.setPositiveButton("Sim"){dialog, which ->
                    val updates = hashMapOf<String,Any>(
                        "nome" to FieldValue.delete(),
                        "idade" to FieldValue.delete(),
                        "bpm" to FieldValue.delete(),
                        "media" to FieldValue.delete()
                    )

colecaoPaciente.document(BLE.nomeBD.toString()).update(updates).addOnSuccessListener
{ Toast.makeText(context, "Campos deletados com sucesso",
Toast.LENGTH_SHORT).show() }
                    colecaoPaciente.document(BLE.nomeBD.toString())
                        .delete()
                        .addOnSuccessListener {
```

```
                        binding.ExEdtNome.setText("")
                        binding.ExTvNome.setText("")
                        binding.ExTvBpm.setText("")
                        binding.ExTvIdade.setText("")
                        binding.ExTvMedia.setText("")

                        Toast.makeText(context, "Paciente deletado com sucesso",
Toast.LENGTH_SHORT).show() }
                    .addOnFailureListener{e-> Toast.makeText(context, "ERRO",
Toast.LENGTH_SHORT).show() }
                binding.ExBtnExcluir.visibility = INVISIBLE

                Toast.makeText(
                    context,
                    "Paciente excluido com sucesso",
                    Toast.LENGTH_SHORT
                )
                    .show()
                getActivity()?.getSupportFragmentManager()!!.popBackStack();
            }

            builder.setNegativeButton("Não"){dialog,which ->
                dialog.dismiss()
                Toast.makeText(requireContext(),"Exclusão
cancelada.",Toast.LENGTH_SHORT).show()
            }

            val dialog: AlertDialog = builder.create()
            dialog.show()
        }
    }
}

    return binding.root
  }

  private fun campoVazio(valor: String): Boolean {
    return valor == "" || valor.isEmpty() || valor.isBlank()
  }

}
```

Apêndice H

Implementação em Kotlin do serviço *BluetoothService*

BluetoothService.kt

```
import android.app.Service
import android.bluetooth.*
import android.content.Intent
import android.nfc.Tag
import android.os.IBinder
import android.util.Log
import com.example.tcc2.form.tcc.principal.*
import com.example.tcc2.form.tcc.util.BLE.bluetoothAdapter
import com.example.tcc2.form.tcc.util.BLE.bluetoothDevice
import com.example.tcc2.form.tcc.util.BLE.bluetoothGatt
import com.example.tcc2.form.tcc.util.BLE.deviceAddress
import com.google.firebase.firestore.FirebaseFirestore
import java.util.*

//Alterado por Gabriel de Paiva no dia 11/08/2022

class BluetoothService : Service() {


    private var timer = Timer()

    override fun onBind(intent: Intent?): IBinder? = null

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        startService()
        return START_REDELIVER_INTENT
    }


    override fun onDestroy() {
        super.onDestroy()
        stopService()
    }

    private fun startService() {
        Log.e(TAG, "startService")
        bluetoothDevice = bluetoothAdapter?.getRemoteDevice(deviceAddress)
        if (bluetoothDevice != null) {
            bluetoothGatt = bluetoothDevice!!.connectGatt(this, true, gattCallback)
        }
        startHeartRate()
    }


    private fun stopService() {
        Log.e(TAG, "stopService")
```

```
        closeGatt()
        stopHeartRate()
        stopSelf()
    }

    private fun closeGatt() {
        bluetoothGatt?.close()
        bluetoothGatt = null
    }

    private fun startHeartRate() {
        timer = Timer()
        timer.schedule(object : TimerTask() {
            override fun run() {
                scanHeartRate()
            }
        }, 0, 1000)

    }

    private fun stopHeartRate() {
        timer.cancel()
    }

    private fun scanHeartRate() {
        if (bluetoothGatt != null) {
            val bluetoothCharacteristic =
bluetoothGatt?.getService(UUIDs.HEART_RATE_SERVICE)
                ?.getCharacteristic(UUIDs.HEART_RATE_CONTROL_CHARACTERISTIC)

            if (bluetoothCharacteristic != null) {
                bluetoothCharacteristic.value = byteArrayOf(21, 1, 1)
                bluetoothGatt!!.writeCharacteristic(bluetoothCharacteristic)


            }
        }
    }

    private fun listenHeartRate() {
        if (bluetoothGatt != null) {
            val bluetoothCharacteristic =
bluetoothGatt!!.getService(UUIDs.HEART_RATE_SERVICE)

.getCharacteristic(UUIDs.HEART_RATE_MEASUREMENT_CHARACTERISTIC)
            bluetoothGatt!!.setCharacteristicNotification(bluetoothCharacteristic, true)
            val descriptor =
bluetoothCharacteristic.getDescriptor(UUIDs.HEART_RATE_DESCRIPTOR)
```

```
            descriptor.value = BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE
            bluetoothGatt!!.writeDescriptor(descriptor)
        }
    }

    private fun broadcastUpdate(action: String) {
        val intent = Intent(action)
        sendBroadcast(intent)
    }

    private fun broadcastUpdate(action: String, characteristic: BluetoothGattCharacteristic) {
        val intent = Intent(action)
        when (characteristic.uuid) {
            UUIDs.HEART_RATE_MEASUREMENT_CHARACTERISTIC -> {
                val heartRate: Int = characteristic.value[1].toInt()

                intent.putExtra(HEART_RATE, heartRate)
            }
        }
        sendBroadcast(intent)
    }

    private val gattCallback = object : BluetoothGattCallback() {
        override fun onConnectionStateChange(gatt: BluetoothGatt?, status: Int, newState: Int) {
            when (newState) {
                BluetoothProfile.STATE_CONNECTED -> {
                    bluetoothGatt?.discoverServices()
                    broadcastUpdate(ACTION_GATT_CONNECTED)
                    Log.e(TAG, "STATE_CONNECTED")
                }
                BluetoothProfile.STATE_DISCONNECTED -> {
                    broadcastUpdate(ACTION_GATT_DISCONNECTED)
                    Log.e(TAG, "STATE_DISCONNECTED")
                }
            }
        }

        override fun onServicesDiscovered(gatt: BluetoothGatt?, status: Int) {
            when (status) {
                BluetoothGatt.GATT_SUCCESS -> {
                    listenHeartRate()
                    broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED)
                }
                else -> Log.e(TAG, "onServicesDiscovered received: $status")
            }
        }

        override fun onCharacteristicRead(
```

```
            gatt: BluetoothGatt?,
            characteristic: BluetoothGattCharacteristic,
            status: Int
        ) {
            broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic)
        }

        override fun onCharacteristicChanged(
            gatt: BluetoothGatt?,
            characteristic: BluetoothGattCharacteristic
        ) {
            broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic)

        }

    }

    companion object {
        private val TAG = BluetoothService::class.java.simpleName
    }

}
```

Apêndice I

Implementação em Kotlin do *UUIDs* utilizados no *BluetoothService.kt*

```
//Alterado por Gabriel de Paiva no dia 11/08/2022


import java.util.UUID

object UUIDs {
    val HEART_RATE_SERVICE: UUID = UUID.fromString("0000180d-0000-1000-8000-
00805f9b34fb")
    val HEART_RATE_MEASUREMENT_CHARACTERISTIC: UUID =
UUID.fromString("00002a37-0000-1000-8000-00805f9b34fb")

    val HEART_RATE_DESCRIPTOR: UUID = UUID.fromString("00002902-0000-1000-
8000-00805f9b34fb")
    val HEART_RATE_CONTROL_CHARACTERISTIC: UUID =
UUID.fromString("00002a39-0000-1000-8000-00805f9b34fb")
```

Apêndice J

Implementação em Kotlin dos objetos de passagem de fragmentos utilizados

BLE.kt

Alterado por Gabriel de Paiva no dia 11/08/2022

```
import android.bluetooth.BluetoothAdapter
import android.bluetooth.BluetoothDevice
import android.bluetooth.BluetoothGatt

object BLE {

    var bluetoothAdapter:BluetoothAdapter = BluetoothAdapter.getDefaultAdapter()
    var bluetoothGatt: BluetoothGatt? = null
    var bluetoothDevice: BluetoothDevice? = null
    var deviceAddress: String? = null

    var bpm: Int? = null
    var nomeBD: String? = null
    var media: Float? =null
    var authUid:String?=null
    var wearableDeviceName: String? = null

}
```

Apêndice K

Licença Apache 2.0

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,

and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the

editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of

the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify

the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing

the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
GABINETE DO REITOR

Av. Universitária, 1069 • Setor Universitário
Caixa Postal 86 • CEP 74605-010
Goiânia • Goiás • Brasil
Fone: (62) 3946.1000
www.pucgoias.edu.br • reitoria@pucgoias.edu.br

RESOLUÇÃO n° 038/2020 – CEPE

ANEXO I

APÊNDICE ao TCC

Termo de autorização de publicação de produção acadêmica

O(A) estudante Gabriel de Paiva Castro do Curso de Engenharia de Computação, matrícula 2016.1.0033.0333-8, telefone: (62) 99624-3989 e-mail gabrielnrg15@hotmail.com, na qualidade de titular dos direitos autorais, em consonância com a Lei nº 9.610/98 (Lei dos Direitos do Autor), autoriza a Pontifícia Universidade Católica de Goiás (PUC Goiás) a disponibilizar o Trabalho de Conclusão de Curso intitulado Monitoramento Personalizado para Alerta de Emergências Utilizando Smartwatch, gratuitamente, sem ressarcimento dos direitos autorais, por 5 (cinco) anos, conforme permissões do documento, em meio eletrônico, na rede mundial de computadores, no formato especificado (Texto(PDF); Imagem (GIF ou JPEG); Som (WAVE, MPEG, AIFF, SND); Vídeo (MPEG, MWV, AVI, QT); outros, específicos da área; para fins de leitura e/ou impressão pela internet, a título de divulgação da produção científica gerada nos cursos de graduação da PUC Goiás.

Goiânia, 29 de setembro de 2022.

Assinatura do autor: Gabriel de Paiva Castro

Nome completo do autor: Gabriel de Paiva Castro

Assinatura do professor–orientador:

Nome completo do professor-orientador: Marcelo Antonio Adad de Araujo