PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Uma Graphical User Interface em Flutter para Monitoramento de dados fisiológicos e a geração de alertas para Pacientes

Mariela Mendonça de Andrade

PROJETO FINAL DE GRADUAÇÃO

CENTRO TÉCNICO CIENTÍFICO - CTC

DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação em Ciência da Computação

Rio de Janeiro, dezembro de 2021.

Mariela Mendonça de Andrade

Uma Graphical User Interface em Flutter para Monitoramento de dados fisiológicos e a geração de alertas para Pacientes

Relatório de Projeto Final, apresentado ao programa de Ciência da Computação da PUC-Rio como requisito para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Markus Endler

Departamento de Informática

Rio de Janeiro dezembro de 2021.

Agradecimentos

A minha família, aos meus amigos que fiz ao longo da faculdade e, por fim, ao professor Markus Endler e Antonio Iyda pelo apoio e orientação que possibilitou o desenvolvimento deste projeto.

Resumo

Este projeto trata da criação de uma aplicação mobile, utilizando o framework Flutter, que tem o objetivo de monitorar pacientes de forma remota de modo que médicos, enfermeiros e/ou cuidadores consigam saber se ocorreram modificações no quadro clínico da pessoa monitorada. O aplicativo será uma interface front-end de um sistema de monitoramento de pacientes que possui as seguintes características: coleta de dados de pacientes em casa ou no hospital, visualização em tempo real de dados do estado clínico do paciente e um sistema de avisos e alarmes gerados pelo sistema.

Palavras-chave: Monitoramento Remoto de Pacientes; Aplicação Mobile; Visualização de dados; IoT; Cuidados Hospitalares.

Abstract

This project deals with the creation of a mobile application, using the Flutter framework, which aims to monitor inpatients remotely so that doctors, nurses and/or caregivers can know if there have been changes in the clinical condition of the monitored person. The application will be a front-end interface to an inpatient monitoring system that has the following features: collection of inpatient data at home or in the hospital, real-time visualization of inpatient clinical status data, and a warning and alarm system generated by the system.

Keywords: Remote Inpatient monitoring; Mobile Application; Data Visualization; IoT; Inpatient care.

Sumário

Lista de Figuras

1 Introdução

No cenário atual, devido à pandemia de COVID-19, o aumento do fluxo de pessoas nos hospitais cresceu consideravelmente. Um exemplo disso, foi a rede hospitalar estadual do Rio de Janeiro ter chegado a taxa de 100% [1] de leitos destinados à doença ocupados no auge da crise sanitária.

Devido a pandemia, a população teve que adaptar suas rotinas, na medida do possível, de modo que não houvesse aglomerações nas ruas, e consequentemente, menos exposição das pessoas ao vírus. Parte desta adaptação só foi possível devido a tecnologia: aulas e trabalho tiveram que ser feitos de forma remota, compras de supermercado e farmácia foram facilitados por meio de aplicativos de entrega, o avanço da telemedicina, entre outros. Além disso, com a diminuição de eventos sociais, a tecnologia foi essencial para manter o contato social e diminuir os efeitos do isolamento nas pessoas [2].

Com o isolamento e o maior contato diário com a tecnologia, os brasileiros passaram a ficar cada vez mais conectados. Entre julho e setembro de 2020, a venda de celulares cresceu cerca de 10% [3] em relação ao mesmo período do ano anterior e foram vendidos cerca de 13,4 milhões de smartphones somente no terceiro trimestre do ano passado. Consequentemente, o aumento de vendas de aparelhos fez com que houvesse um aumento no número de acesso móvel à internet, registrando em dezembro de 2020 mais de 234 milhões de acessos [4].

A facilitação do acesso a smartphones e redes móveis de internet tornou o consumo de informações mais rápido e praticamente instantâneo. Pensando nisso, o propósito do sistema deste trabalho é usar essa conectividade e aplicá-la no monitoramento de paciente em tempo real.

Este projeto é extensão do Projeto do Sistema Inteligente de Monitoramento Paramétrico de Pacientes em enfermarias – COVID-19 (SH-Sens) [5] e funcionará como uma interface front-end de um sistema de monitoramento de pacientes. Esta interface será desenvolvida como aplicativo mobile, estando disponível nos principais sistemas operacionais utilizados atualmente. Além disso, será disponibilizada uma versão web da interface, para que assim, o seu uso possa ser ainda mais ampliado. O ambiente tecnológico de desenvolvimento utilizado para este trabalho foi o Windows 10 e o MacOS 11.4 e as plataformas tecnológicas utilizadas foram os dispositivos mobile (sistemas iOS e Android) e o dispositivo web Chrome. Este projeto é programado utilizando a linguagem Dart e o framework Flutter.

2 A Situação atual

A popularidade da internet das coisas (IoT) está aumentando a cada ano. Sua aplicação e estudo pode ser vista em casas inteligentes [6] que trazem praticidade, economia de energia e segurança ou até mesmo na detecção de poluição da água [7], por meio de uma rede de esgoto inteligentes que ajudam a proteger os rios e o seu ecossistema. Além disso, este assunto também é bastante popular na área da saúde, principalmente na questão de monitoramento remoto de pacientes.

Com o monitoramento remoto [8], pacientes podem evitar uma ida ao hospital, evitando assim o contato com pessoas que podem estar contaminadas com o vírus da COVID-19, além de melhorar a qualidade de vida do paciente pois evita o deslocamento desnecessário a unidade de saúde, sendo um diferencial para pessoas com problemas de mobilidade.

Em um estudo da universidade de Stanford [9], foi descoberto que o Apple Watch pode ser usado para medir, via aplicativo e sensores, uma versão do teste clínico de caminhada dos seis minutos [10], que tem como principal objetivo descobrir a capacidade respiratória, cardíaca e metabólica que um paciente possui. Nos testes realizados com supervisão, o teste via aplicativo teve uma sensibilidade de 90% e taxas de especificidade de 85% ao avaliar a fragilidade cardiovascular de um paciente, enquanto em testes conduzidos sem supervisão tiveram uma sensibilidade de 83% e 60% respectivamente. Entretanto, apesar da diferença entre as porcentagens, os especialistas afirmam que os resultados foram altos o suficiente para serem considerados efetivos.

Além disso, a operadora de saúde americana Geisinger Health Plan (GHP) [11], realizou um estudo no qual foram monitorados remotamente durante 70 meses um grupo de 541 pacientes acima de 65 anos com problemas cardíacos. Nos primeiros 30 dias, houve uma queda de 44% na admissão hospitalar e que esta taxa após 90 dias se estabilizou em 38%. Segundo a GHP, entre 2008 e 2014 houve uma economia de 11% ao mês por paciente devido ao monitoramento remoto.

Como vimos acima, além do monitoramento remoto ser mais econômico e prático, ele pode ser bastante eficiente para a equipe médica. De acordo com um projeto feito no hospital Patong [12], na Tailândia, o monitoramento e anotação de sinais vitais de pacientes era feito de 4 em 4 horas de forma totalmente manual, no qual a equipe médica preenchia formulários com os dados dos pacientes. O principal problema do acompanhamento manual de sinais vitais de

pacientes é a comunicação, uma vez que, uma simples rasura no formulário ou alguma escrita confusa compromete totalmente o acompanhamento daquele paciente. A avaliação do sistema mostrou que a solução proposta pode monitorar com precisão os sinais vitais e parâmetros de monitoramento adicionais e que pode reduzir a carga de trabalho de enfermeiras e equipes de apoio.

3 Objetivos do trabalho

O objetivo deste trabalho para mim, como aluna, seria aplicar os conhecimentos que foram aprendidos durante toda a minha graduação na implementação de uma interface mobile, que usasse técnicas modernas de internet das coisas e que utilizasse a minha experiência com o desenvolvimento de aplicativos iOS. Já o objetivo do projeto, é fazer uma interface para médicos e pacientes de um sistema de monitoramento, de forma que fosse leve, amigável e prático.

Esta interface possui as seguintes propriedades: coleta de dados de pacientes em casa ou no hospital, visualização em tempo real de dados do estado clínico do paciente, inserção de sintomas não detectados por sensores e geração de avisos e alarmes realizados pelo sistema.

Sendo assim, o aplicativo desenvolvido deverá que conter as seguintes características:

1. Diferentes interfaces para os diferentes casos de uso propostos.
2. Alertas sonoros e físicos de recebimento de alarmes que podem ser desabilitados pelo usuário.
3. Conformidade com a Lei Geral de Proteção de Dados (LGPD).
4. Inserção de dados não monitorados, de acordo com o protocolo National Early Warning Score 2 (NEWS 2) [13].
5. Histórico de alertas da enfermaria e de uma determinada cama.
6. Aplicativo desenvolvido nos principais sistemas operacionais mobile (Android e iOS) e em WebApp.
7. Aviso de desconexão com os sensores.

4 Fundamentos

A Figura 1 descreve o modelo conceitual do Projeto do Sistema Inteligente de Monitoramento Paramétrico de Pacientes em enfermarias – COVID-19 (SH-Sens) [5].

Este projeto tem como objetivo geral *"fornecer solução escalável de monitoramento permanente remoto de pacientes de COVID-19 para aprimorar o gerenciamento do atendimento e acompanhamento das equipes de saúde, principalmente no sentido de identificar alterações no quadro clínico destes pacientes." (PAGANELLI, 2020, p. 5).*

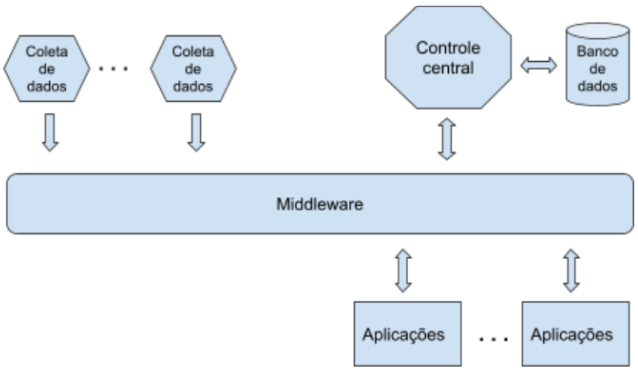

Figura 1. Modelo conceitual do sistema SH-Sens [5].

O modelo apresentado contém as seguintes características:

- Camada *Middleware:* é responsável pela integração com os demais módulos do sistema. Ela foi implementada usando o protocolo MQTT, e por isso, todas as interações entre módulos utilizam este padrão.
- Camada Coleta de dados: representa a leitura e envio dos dados dos sensores aos demais elementos.
- Camada Controle Central: é responsável pelo processamento e armazenamento de dados, podendo ser considerado como o *backend* do sistema.
- Camada Aplicações: representa a interface com o usuário, ou o *frontend* do sistema. É nesta camada na qual este projeto se encaixa no sistema como um todo.

A aplicação mobile irá se comunicar com as demais camadas por meio de *Application Programming Interfaces* (APIs) que deverão ser estendidas e adaptadas para suportar todos os casos de uso que serão implementados.

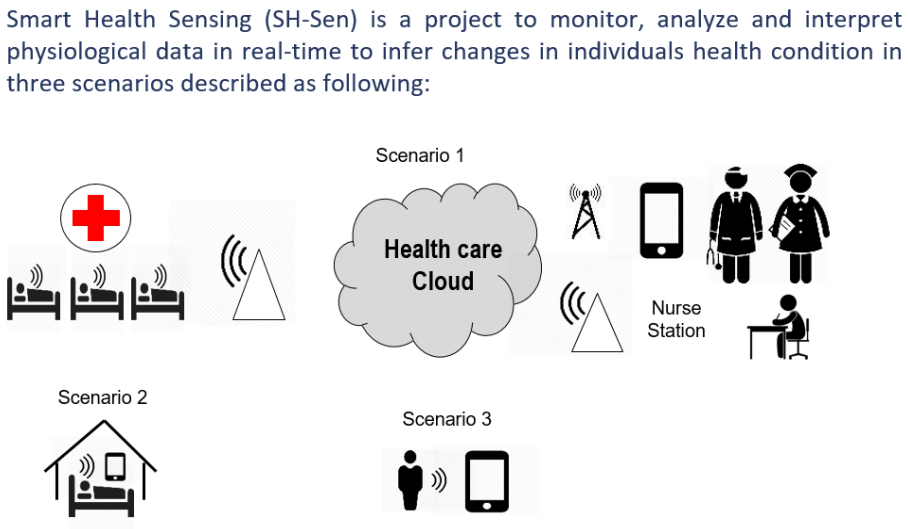

Figura 2. Casos de uso do sistema.

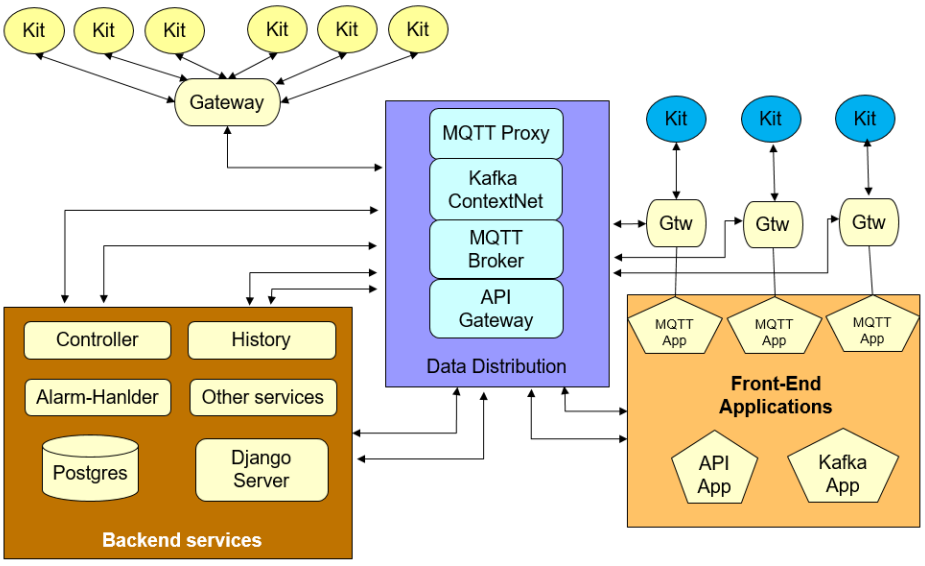

Figura 3. Interações do aplicativo com o sistema.

Na Figura 3 temos uma noção melhor de como será a interação do aplicativo no sistema. Nela, o App é visto como uma aplicação *frontend* e é representado pelo componente *MQTT App*. A aplicação mobile também atuará

como um gateway entre os sensores e as aplicações do *backend*, ou seja, o SH-Sens prevê um processo *gateway* rodando no mobile. Com isso, o aplicativo poderá ter uma conexão direta com o *gateway* para receber os dados dos biossensores, assim como também poderá receber informações analisadas pelo *backend* através de uma conexão de internet em paralelo.

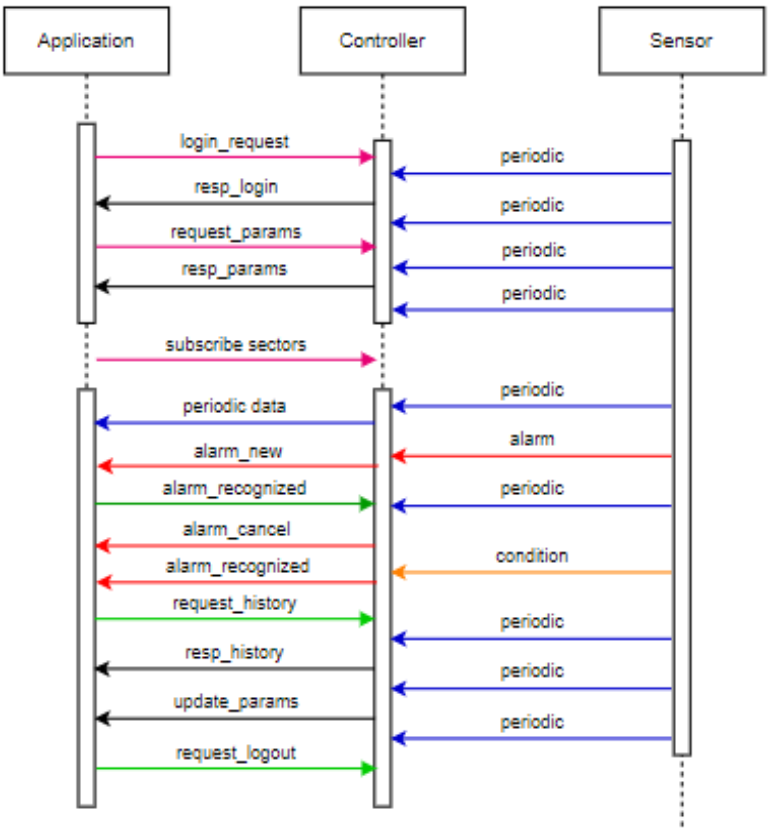

Figura 4. Diagrama de Sequência do sistema SH-Sens[5].

Na Figura 4 temos o diagrama de sequência que representa a comunicação entre os elementos do sistema desde o login.

É suposto que o usuário já tenha feito o cadastro no *backend* da aplicação, e com isso, já tenha aceitado o termo de uso de seus dados. Caso o login seja bem sucedido, será feita uma requisição dos parâmetros iniciais da aplicação e o controlador retornará estas informações. A aplicação então fará uma requisição para se subscrever nos setores, para que assim ela consiga receber as atualizações dos mesmos, seus parâmetros (como por exemplo o estado do leito) e reconhecer alarmes. Poderá também ser feita a solicitação do histórico de um paciente ou leito e, além disso, a aplicação poderá receber uma

atualização de algum parâmetro inicial, como por exemplo no caso de algum leito ser liberado.

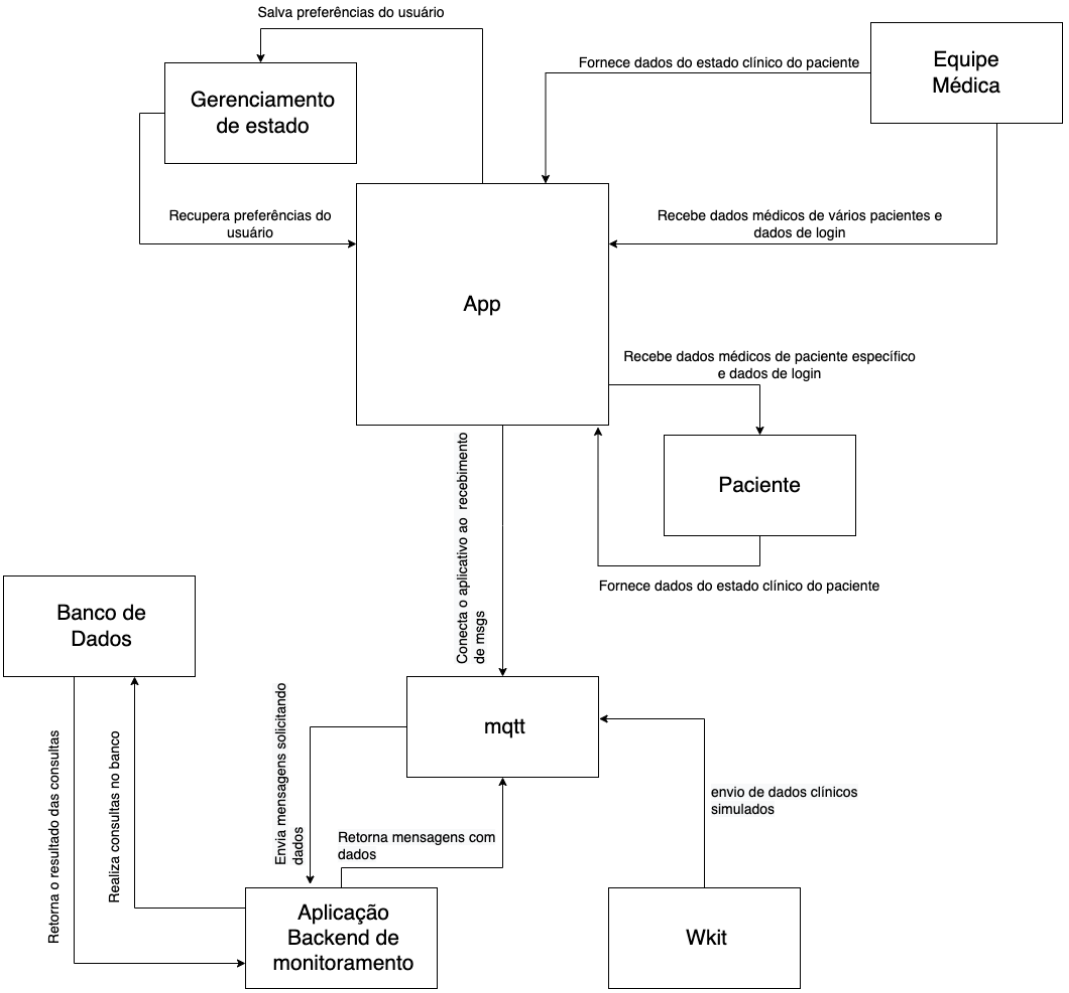

Figura 5. Fluxo de dados do Projeto Mobile Desenvolvido.

O diagrama acima, Figura 5, consegue mostrar de uma forma geral, como está sendo realizado o fluxo de dados do aplicativo proposto neste projeto. Os dados recebidos pela aplicação são gerados pelo Wkit Simulator, cujo objetivo é simular dados de um sensor que estaria conectado a um paciente.

O aplicativo se conecta ao *backend* por meio do protocolo MQTT [14], que é *"um protocolo de mensagens leve para sensores e pequenos dispositivos móveis otimizado para redes TCP/IP." (MQTT, 2021).* Nele, a aplicação se subscreve nos tópicos necessários para continuar o seu fluxo. Além disso, dependendo de qual seja o cenário de uso do aplicativo, informações diferentes são recebidas e exibidas para o usuário: No caso de uso da equipe médica,

haverá um dashboard com todas as camas do setor monitorado. Já no caso de uso do paciente/cuidador, será exibido somente os dados daquela pessoa monitorada.

4.1 O MQTT

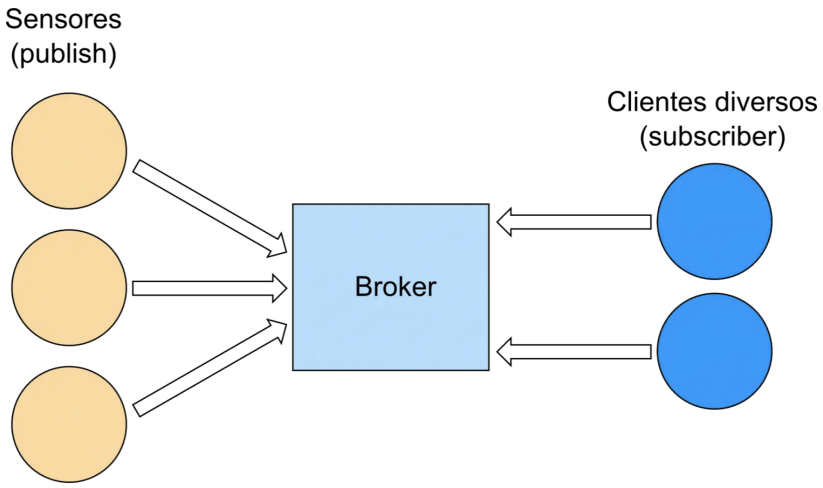

Figura 6. Overview do MQTT[15].

Conforme mencionado na seção acima, o aplicativo desenvolvido neste projeto se conecta ao *backend*, por meio do Protocolo MQTT. O *backend* possui as APIs para conexão com o banco de dados e configurações essenciais para o funcionamento da aplicação.

As trocas de mensagens realizadas por meio deste protocolo seguem o padrão *publish/subscribe*, ou seja, caso seja necessário receber uma informação específica, é preciso se subscrever no tópico que contenha aquela informação. Esta subscrição é feita através do broker, que é um intermediário entre os dados dos sensores e o cliente, conforme ilustrado na Figura 6.

4.2 O Flutter

Flutter [16] é um *UI toolkit open source*, ou seja, é um kit de ferramentas de interface do usuário, desenvolvido pelo Google, e que permite a reutilização de código por diversos sistemas operacionais tais como Android, iOS, Web e Desktop.

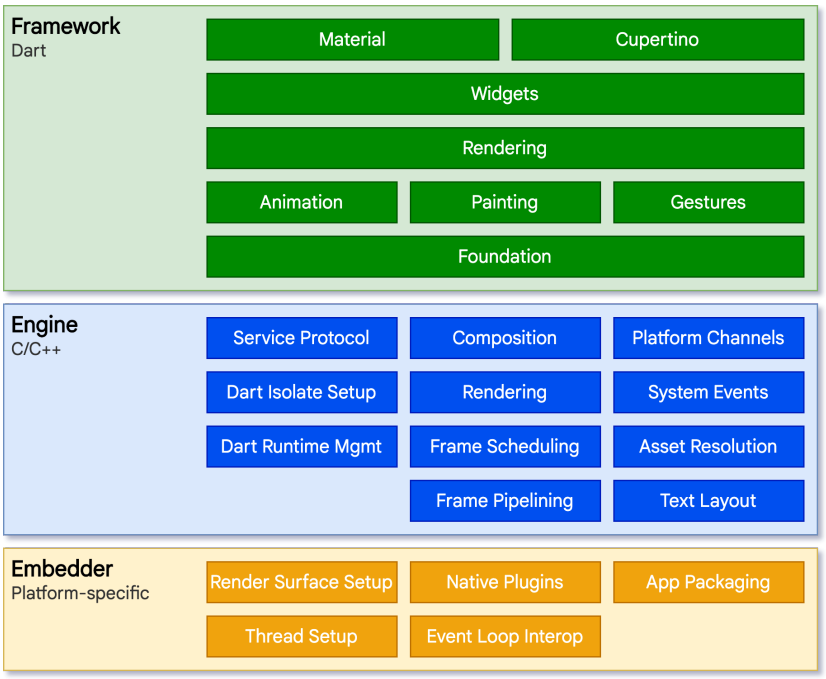

Figura 7. Arquitetura do framework Flutter.

Seu principal objetivo [17] é permitir que os desenvolvedores entreguem aplicativos de alta performance que pareçam naturais em diversas plataformas, levando em consideração as suas diferenças nativas e possuindo compartilhamento de código no desenvolvimento multiplataforma.

A Figura 7 representa a arquitetura do Flutter, que possui basicamente três camadas: a primeira delas é a camada *Embedder* tem a função de fornecer um ponto de entrada entre a aplicação e o sistema operacional, sendo por ele que serviços como superfície de renderização, acessibilidade e input são acessados. A segunda camada é a *Engine* que dá o suporte básico de ações como por exemplo renderização, composição e eventos do sistema para as aplicações Flutter. A última camada é o *Framework,* que é a camada que o desenvolvedor terá contato e irá interagir através da linguagem Dart.

Nesta última camada, os desenvolvedores tem acesso aos principais componentes utilizados no desenvolvimento de aplicativos, tais como: animações, gestos, painting [18], que são os mecanismos responsáveis por aplicar sombras, escala de imagens e bordas de contorno. Além disso, temos os Widgets que podem ser separados em dois tipos: Material e Cupertino. Os widgets Material são aqueles pertencentes ao *Material Design*, guia de estilo desenvolvido pelo Google [19], já os widgets Cupertino são aqueles pertencentes ao guia de estilo desenvolvido pela Apple [20].

No desenvolvimento mobile, é comum que existam APIs específicas para cada plataforma e pensando nisso, o Flutter permite que pedaços de códigos customizados possam ser utilizados por meio de *platform channel*, que é uma maneira de comunicar o seu código híbrido com o código específico de uma plataforma.

Apesar da facilidade de conseguir desenvolver aplicativos para diferentes plataformas com máximo de compartilhamento de código possível, é natural que cada uma destas plataformas tenha componentes visuais ou gestos que sejam específicos de cada uma delas e é importante que o aplicativo respeite os padrões do iOS e do Android.

Como os sistemas possuem alguns componentes diferentes, é natural que ao realizar o *layout* da tela, essa diferenciação seja feita. Na Figura 8, temos um exemplo de carregamento das informações na tela: Caso as informações ainda não tenham sido carregadas, o indicador de carregamento é exibido ao usuário. Para isso, verificamos qual sistema o usuário está utilizando e, de acordo com ele, exibimos o componente adequado. A Figura 9 exemplifica como que este componente é diferente na prática.

```
if (_platform == TargetPlatform.iOS) {
  return Center(child: CupertinoActivityIndicator());
} else {
  return Center(child: CircularProgressIndicator());
}
```

Figura 8. Exemplo de uso dos componentes de acordo com o sistema operacional.

Figura 9. Exemplo dos componentes diferentes: à esquerda, o componente Android e à direita o componente iOS.

5 Atividades Realizadas

5.1 A escolha do Framework

Existem diversas linguagens no mercado voltadas para o desenvolvimento de aplicativos. Dentre elas quatro são as mais utilizadas pelos desenvolvedores, que são:

- As linguagens nativas Kotlin e Swift, sendo usadas para o desenvolvimento de aplicativos nos sistemas Android e iOS respectivamente.
- As linguagens híbridas Flutter e React Native, que podem ser utilizadas para o desenvolvimento em diversos dispositivos.

Nos referimos como linguagens nativas, aquelas que possuem como finalidade uma plataforma específica. Já as linguagens que são consideradas híbridas, geralmente possuem diversas formas de aplicações, sendo uma delas o desenvolvimento de aplicativos.

A escolha pelas linguagens nativas possui a vantagem de ter maior performance e de possuírem gestos e movimentos que já são do conhecimento dos usuários de determinado sistema operacional. Entretanto, a escolha por essas linguagens demanda um maior tempo de desenvolvimento de um aplicativo multiplataforma, uma vez que para cada plataforma será um código diferente e requer uma equipe especializada nas plataformas que serão utilizadas.

A escolha pelas linguagens híbridas possui a vantagem de ter seu tempo de desenvolvimento reduzido e por geralmente possuir uma curva de aprendizagem mais rápida, por serem programados em linguagens populares como por exemplo o Javascript. Entretanto, dificilmente todas as funcionalidades desenhadas para esse aplicativo terão um desempenho idêntico em ambas as plataformas, podendo afetar a experiência do usuário.

Levando todos estes pontos em consideração, a escolha da linguagem de desenvolvimento baseou-se em dois pontos principais: tempo de desenvolvimento e curva de aprendizagem.

Como o projeto final tem duração de 2 semestres, seria necessário que fosse feita a escolha por uma linguagem híbrida. Por mais que eu tenha bastante experiência em desenvolvimento de aplicativos nativos utilizando o Swift, seria

necessário fazer um estudo na linguagem Kotlin, o que poderia comprometer a entrega do trabalho.

Com a linguagem definida como híbrida, entrou em questão da curva de aprendizado. Eu já havia tido experiência com desenvolvimento Flutter anteriormente, diferentemente da minha situação com o React Native. Além disso, Flutter é um framework open source criado e já está sendo utilizado por diversas empresas como iFood e Nubank.

A minha escolha pelo Flutter se baseou pela experiência com o framework e a oportunidade de poder aprofundar meus conhecimentos nele.

5.2 Tecnologias utilizadas

A principal linguagem de programação utilizada neste projeto foi o Dart, no *frontend*, e o Python, no *backend*. Por já ter tido contato com o Dart anteriormente não tive dificuldades com este requisito no desenvolvimento do projeto. Entretanto, trabalhar com o Python e o *backend* foi algo que eu nunca tinha feito antes e tive algumas dificuldades no início para entender como os dados eram relacionados.

O ambiente tecnológico de desenvolvimento utilizado para este trabalho foi Visual Code, Android Studio e XCode para o desenvolvimento da aplicação mobile e web. Para o *backend*, foi utilizado o Visual Code e o Docker. Apesar de já ter experiência com o desenvolvimento mobile, eu nunca havia trabalhado em um cenário no qual o *backend* estivesse funcionando em containers, portanto tive que estudar e pesquisar sobre isso antes de iniciar o projeto.

Já em relação ao protocolo MQTT, eu nunca havia trabalhado ou conhecido ele antes do projeto. Foi sem dúvidas a parte mais complicada do trabalho, visto que não se possui muitas informações online sobre o uso deste protocolo com dispositivos programados em Flutter, sendo que muitas vezes eu tive que trabalhar fazendo comparações entre as linguagens dos tutoriais e o Dart. Entretanto, o ponto positivo de trabalhar com este protocolo foi poder fazer o envio de mensagens de forma contínua sem que fossem necessárias abrir várias sessões, como se é feito ao utilizar API Rest.

5.3 Execução do WKit Simulator e do SmartAlarmsApp

Para que fosse feita a integração deste projeto com o SH-SENS, foi necessário clonar o repositório do *wkit-simulator*. Nele, há toda a configuração do *backend* necessário para esta aplicação, além das imagens dos containers e das APIs utilizadas.

Com o projeto baixado, e com o cliente Docker instalado no computador, basta seguir as instruções de instalação contidas no readme do repositório do *wkit-simulator*. Uma vez tendo o ambiente funcionando, Figura 10, é necessário criar os sensores, pacientes, camas, login, e setores dentro do container django para que possam ser gerados os dados usados no aplicativo.

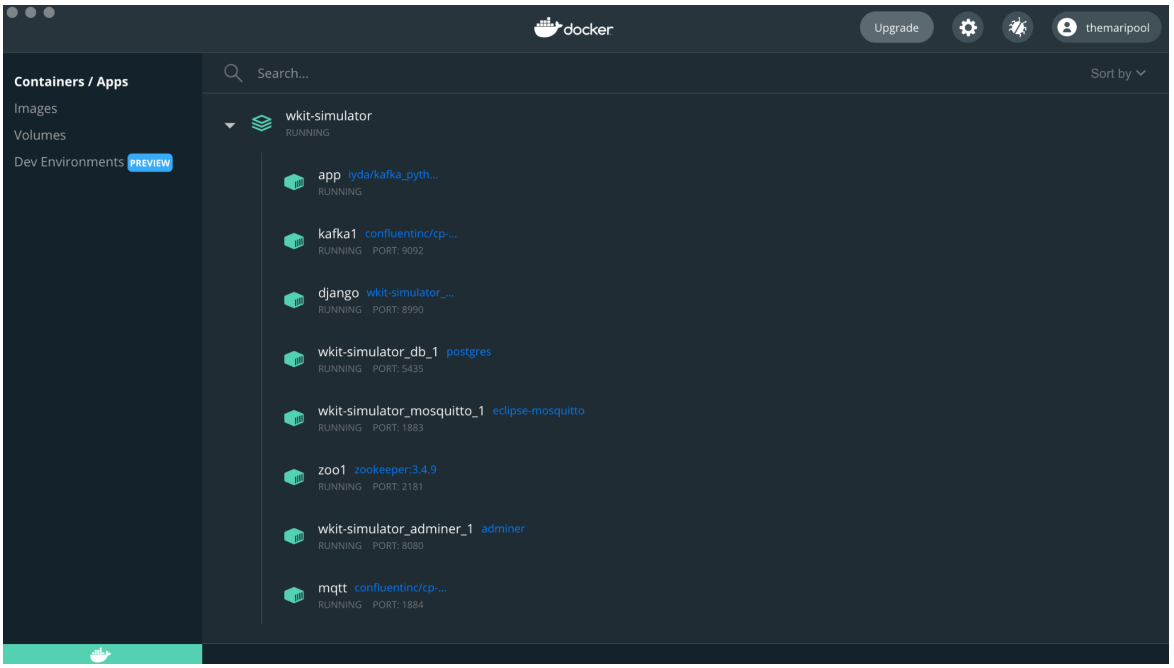

Figura 10. Containers da aplicação.

Em seguida, basta executar o simulador de dados dos sensores, dentro do container app, e buildar o aplicativo, conforme ilustrado na Figura 11. O login utilizado no aplicativo deve ser o mesmo que foi criado na inserção dos dados dos pacientes.

```
# bash
root@docker-desktop:/django# cd kafkaclientpython/simulators/
root@docker-desktop:/django/kafkaclientpython/simulators# python test_wkit_thread.py
MainThread - test_wkit_thread.py: <module>: Starting ....
MainThread - test_wkit_thread.py: <module>: numero de kits = 4
MainThread - test_wkit_thread.py: <module>: wKit - 1001 - filePath /datasets/Mimic_II/3182539n - db: MIMIC_II
```

Figura 11. Execução do simulador.

5.4 Testes de unidade

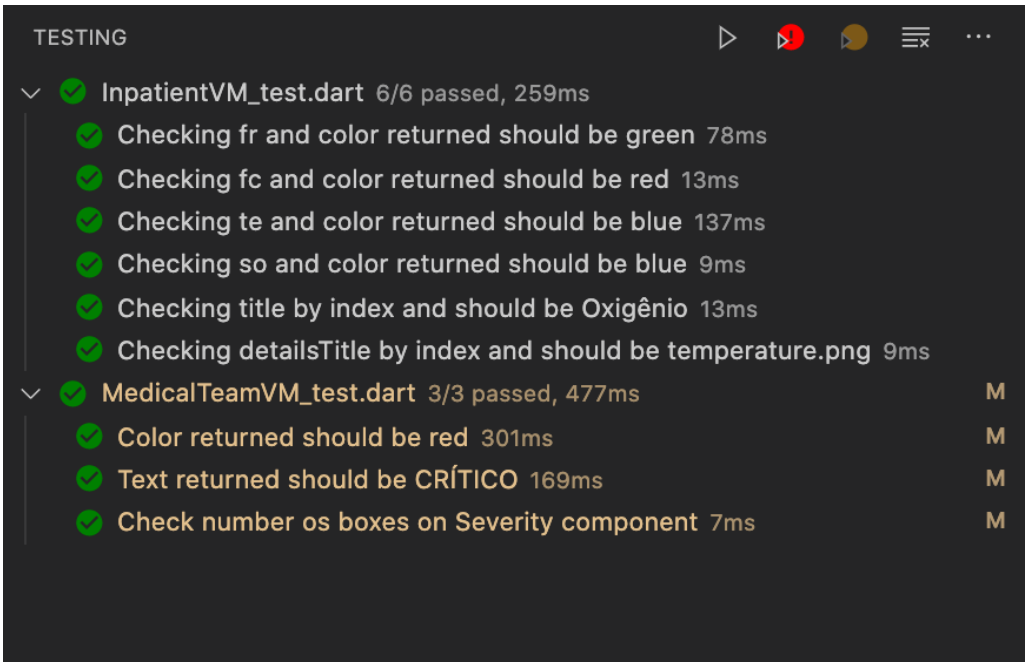

Figura 12. Testes de unidade realizados.

A Figura 12 mostra os testes de unidade que foram realizados em funções do aplicativo que forneciam informações cruciais para a aplicação. Por exemplo, os alertas e os componentes da cama são a parte mais crítica deste projeto, portanto não devem exibir informações ou cores erradas, pois isso iria comprometer a informação do real estado clínico do paciente.

5.5 Método

Este projeto dividiu-se nas seguintes etapas:

1) Estudo aprofundado do framework Flutter.

2) Levantamento dos casos de uso.

3) Levantamento dos artigos relacionados.

4) Implementação visual do caso de uso médico.

5) Preparação e ambientalização com ambiente de desenvolvimento.

6) Estudo da biblioteca MQTT com aplicativos Flutter.

7) Integração do aplicativo com os dados do *backend* via MQTT.

8) Estudo das similaridades e diferenças entre aplicativos e webApps com Flutter.

9) Estudo e Implementação de webApps.

10) Aplicação da Lei de Proteção de Dados em aplicativos.

11) Alterações e extensão de funcionalidades do *backend* para o aplicativo.

12) Estudo e implementação de testes de unidade

Abaixo temos o cronograma final realizado do projeto:

Item	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
1	x	x	x	x	x					
2	x	x								
3	x	x						x	x	
4		x	x	x						
5				x	x	x				
6				x	x	x	x			
7						x	x			
8								x	x	
9								x	x	x
10										x
11									x	x
12								x		

Cabe ressaltar que o plano original deste projeto final contemplava a realização de um teste de usabilidade, entretanto até o fechamento deste relatório não foi possível realizá-los.

6 Implementação do sistema

6.1 O projeto

Na aplicação possuímos dois casos de uso: o caso de uso da equipe médica localizada no hospital e o caso de uso do paciente ou cuidador.

O primeiro caso de uso é focado na equipe médica que está em constante contato com os pacientes internados no hospital, tendo que monitorar diversas camas ao longo do plantão. Já o segundo caso é quando a pessoa monitorada não está mais no hospital, porém, necessita ter o seu estado clínico

acompanhado de forma constante. Neste caso, o próprio paciente pode acompanhar os seus dados vitais ou este acompanhamento pode ser feito por um cuidador.

National Early Warning Score 2 (NEWS 2) – versão brasileira

Parâmetros Fisiológicos	Pontuação						
	3	2	1	0	1	2	3
Frequência respiratória (por minuto)	≤8		9-11	12-20		21-24	≥25
Sp02 % - Escala 1	≤91	92-93	94-95	≥96			
Sp02 % - Escala 2	≤83	84-85	86-87	88-92 ≥93 em ar ambiente	93-94 com oxigênio	95-96 com oxigênio	≥97 com oxigênio
Ar ambiente ou oxigênio?		Oxigênio		Ar Ambiente			
Pressão arterial sistólica(mmHg)	≤90	91-100	101-110	111-219			≥220
Pulso (por minuto)	≤40		41-50	51-90	91-110	111-130	≥131
Consciência				Alerta			Confusão aguda Resposta a voz ou dor Irresposivo
Temperatura (°C)	≤35.0		35.1-36.0	36.1-38.0	38.1-39.0	≥39.1	

National Early Warning Score 2 (NEWS 2) ©Royal College Of Physicians 2017. Adaptação transcultural para português. Brasil, 2018.

Figura 13. Tabela de pontuação do NEWS 2 - versão traduzida.

O monitoramento dos pacientes, feito pelos sensores, segue um sistema de pontuação de severidade conhecido como National Early Warning Score 2 (NEWS 2) [21], ilustrado na Figura 13. Este sistema tem o objetivo de padronizar a forma de avaliar e responder a doenças que possuem uma mudança de quadro rápida, como é o exemplo da COVID-19.

Este protocolo consiste em atribuir uma pontuação para os dados fisiológicos que são monitorados. São monitorados seis dados: frequência respiratória, saturação, pressão arterial, pulso, consciência e temperatura. Cada dado possui uma variação máxima que está associada a uma pontuação menos grave (0) até o estado mais crítico (3).

Os valores de cada parâmetro são somados e, caso o paciente necessite de suplementação de oxigênio, 2 pontos extras são adicionados. Com o resultado em mão, os médicos podem agir de acordo com a pontuação total. A Figura 14 exemplifica quais são os planos de ação para cada caso.

Score NEW	Risco clínico	Frequência de monitoramento	Resposta
0 – 4	Baixo	– Mínimo de 12/12 horas se o *score* for 0; – Mínimo de 4/4 ou 6/6 horas se *score* entre 1 e 4.	Avaliação pelo profissional de enfermagem competente ou equivalente, para decidir mudança na frequência do monitoramento ou escala de cuidado.
Score de 3 em qualquer parâmetro individual	Baixo-médio	– Mínimo de 1/1 hora.	Revisão urgente por médico para decidir mudança na frequência do monitoramento clínico ou escala de cuidado.
5 – 6	Médio	– Mínimo de 1/1 hora.	Revisão urgente por médico para decidir mudança na frequência do monitoramento clínico ou escala de cuidado.
≥ 7	Alto	– Monitoramento contínuo dos sinais vitais.	Avaliação emergencial pela equipe clínica ou time de resposta rápida e transferência para cuidados intensivos ou compatíveis / equivalentes.

Figura 14. Tabela de atendimento NEWS de acordo com a pontuação [21].

Segundo o portal PEBMED [22], o sistema NEWS2 *"pode ser considerado um dos melhores instrumentos para avaliação do risco fisiológico de deterioração"* (NEWS2, 2021) e é utilizado em 80% dos hospitais na Inglaterra. Além disso, em um estudo durante a pandemia de Covid-19 [23] utilizando o protocolo na admissão de pacientes no hospital, e com o objetivo de estudar seu desempenho na predição de complicações e mortalidade hospitalar. O resultado do estudo, que contou com a participação de 66 pacientes dos quais 23% desenvolveram alguma doença grave e 20% faleceram, mostrou que o protocolo possui uma sensibilidade de 80% na predição de doenças severas, sendo superior a outros métodos de pontuação de risco usados com o mesmo objetivo.

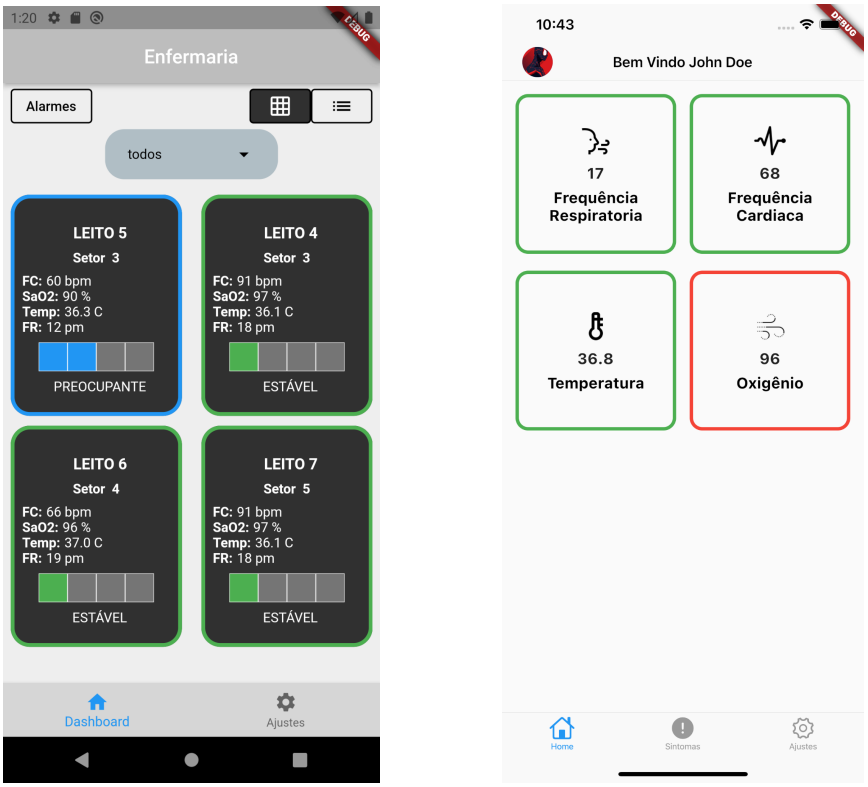

Figura 15. Tela inicial : Caso de uso da equipe médica localizada no hospital vs Caso de uso do paciente ou cuidador.

Ambos os casos de uso são bastante similares, divergindo somente na tela inicial de sua aplicação: enquanto a equipe médica possui uma visão geral de todos os leitos monitorados no qual o usuário está cadastrado, no caso de uso do cuidador, sua tela inicial possui somente os dados monitorados de apenas um paciente.

A tela inicial da aplicação no caso da equipe médica é possível monitorar os dados dos leitos por meio de textos e de componentes visuais.

Figura 16. Componente de severidade da cama - modo tabela.

O componente de severidade da cama, Figura 16, muda de cor de acordo com a mudança da severidade do seu estado clínico. Possuímos quatro níveis, são eles:

1. Estável, representado pela cor verde.
2. Preocupante, representado pela cor azul.
3. Severo, representado pela cor amarela.
4. Crítico, representado pela cor vermelha.

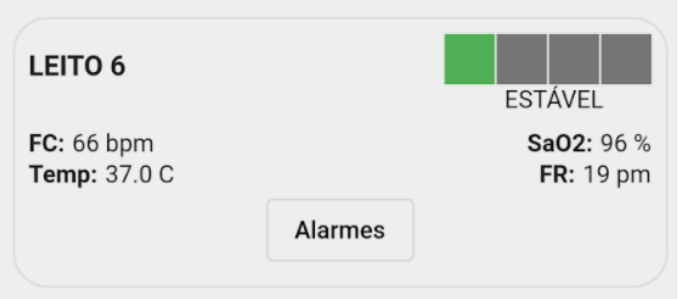

Figura 17. Variação do componente de severidade da cama - modo lista.

Além disso, nesta tela inicial, é possível filtrar as camas por setores para uma melhor visualização de todos os leitos, ver os leitos monitorados em forma de lista, como exemplificado na Figura 17 e acessar uma lista com todos os alertas das camas monitoradas, Figura 18.

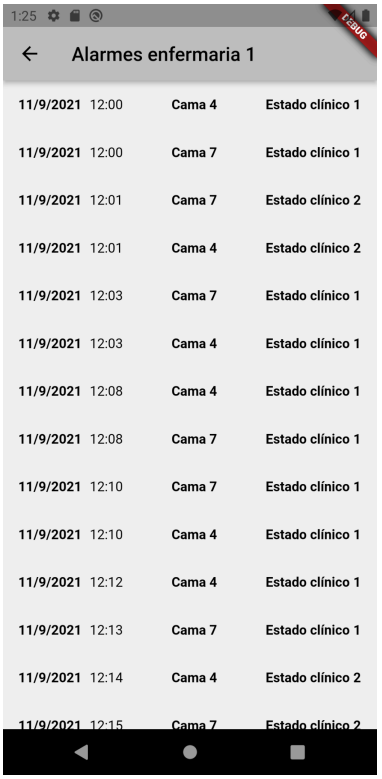

Figura 18. Lista com todos os alertas da enfermaria.

Ao clicar em algum leito, temos acesso a informações mais detalhadas sobre o paciente monitorado em forma de gráfico, como é mostrado na Figura 19. Nele, é mostrada a oscilação dos parâmetros monitorados de acordo com o passar do tempo, fornecendo uma visão mais ampla sobre o estado do paciente. É possível escolher qual dado será exibido no gráfico por meio das seleções dos checkboxes. Além disso, podemos ver somente os alertas daquele leito específico e adicionar sintomas que o paciente esteja sentindo, clicando no símbolo de adição no canto superior direito da tela.

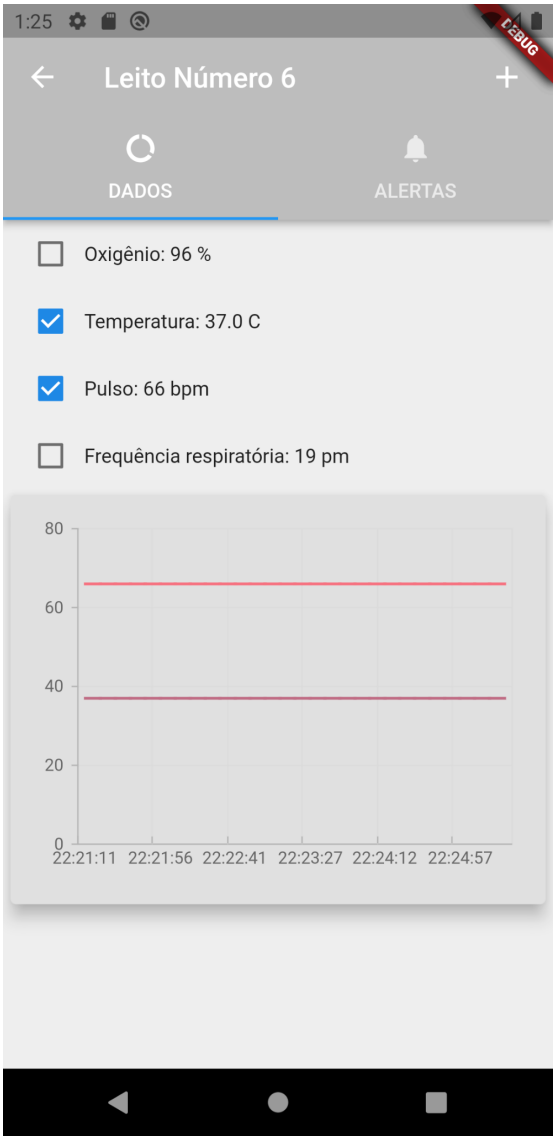

Figura 19. Detalhes do leito monitorado.

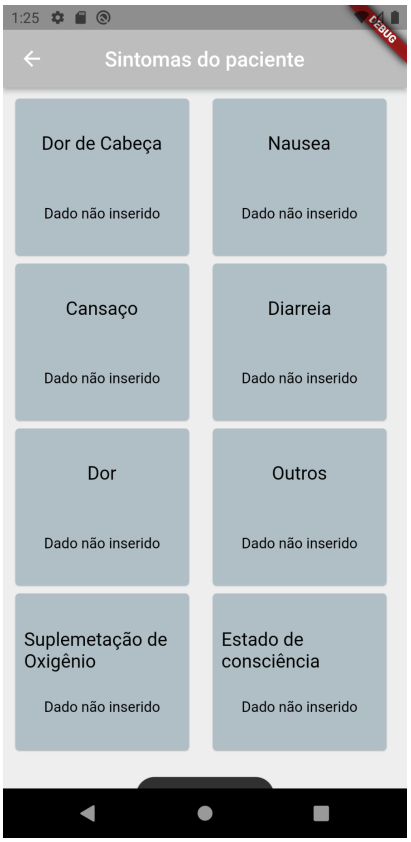

Figura 20. Tela de sintomas.

Para que houvesse um melhor acompanhamento do estado do paciente, foi feita uma tela para fazer o acompanhamento dos dados não captados pelos sensores, ilustrada na Figura 20. São eles: dor de cabeça, náusea, cansaço, diarreia, dor, outros, suplementação de oxigênio e estado de consciência. Estes dois últimos foram inseridos com o intuito de completar os dados coletados pelo NEWS 2 e que não estavam no sistema.

Para inserir um sintoma, basta clicar no componente que indica o dado a ser inserido. Ao clicar, será apresentado um alerta ao usuário no qual ele poderá inserir seu grau de desconforto. Como diferentes tipos de dados podem ser inseridos na tela de sintomas, cada um possui uma classificação diferente: os campos dor de cabeça, náusea, cansaço, diarreia e dor possuem uma classificação de 0 a 5, sendo 0 sem desconforto e 5 o maior nível suportado, conforme ilustrado na Figura 21; O campo oxigênio possui dois botões de sim ou não; O campo estado de consciência possui botões com os diversos níveis que um paciente pode apresentar; O campo outros possui um espaço para que seja inserido uma breve descrição de que outro sintoma está sendo sentido. Nesta tela é possível ver os sintomas previamente inseridos daquele paciente para melhor acompanhamento de seu estado.

Figura 21. Alerta de inserção de dados.

Caso ocorra alguma mudança significativa em algum estado clínico, alertas são exibidos na tela, comunicando via texto e cor, em qual nível de severidade o paciente se encontra. Além do alerta visual, é exibido um alerta via local notification no celular, e caso o usuário tenha habilitado o som no aplicativo, um alerta sonoro também será executado. Caso o usuário não queira este comportamento, ele poderá desabilitar as notificações e som do aplicativo nas configurações do sistema do celular, porém o alerta exemplificado na Figura 22 sempre será exibido caso ocorra mudança no estado do paciente.

Figura 22. Alerta de estado do paciente.

Por fim, os dois casos de uso possuem a tela de ajustes, ilustrada na Figura 23. Nesta tela é possível mudar a imagem de perfil, habilitar o modo escuro e acessar uma pequena descrição sobre o aplicativo.

Além disso, o botão Uso de dados exibe um documento com todos os dados que são coletados ou usados no aplicativo. Em futuras versões da aplicação, seria interessante que o usuário pudesse gerenciar seus dados e ter mais transparência sobre a utilização dos mesmos. Este termo foi gerado automaticamente por um site [24], porém caso a aplicação seja comercialmente lançada será necessário fazer a alteração deste documento.

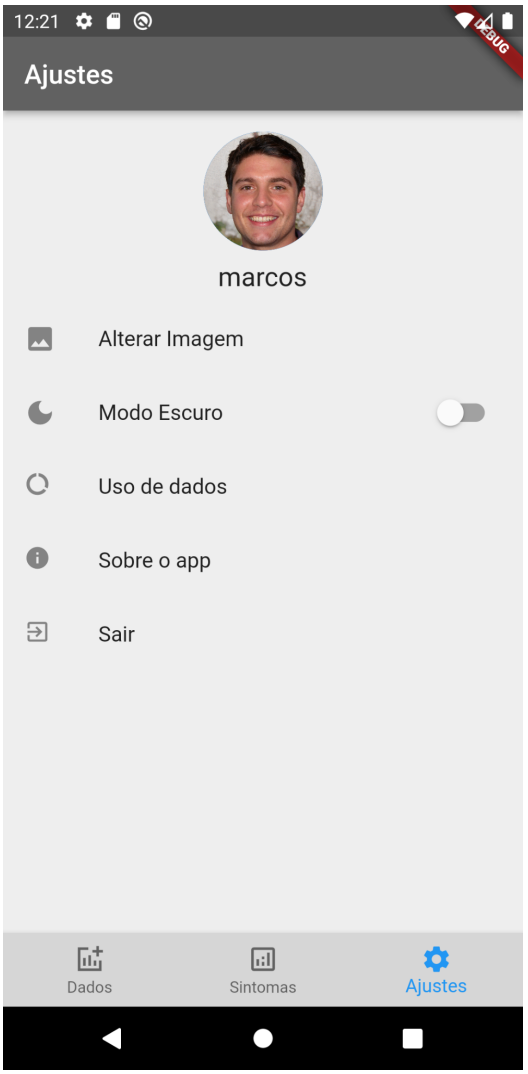

Figura 23. Tela de Ajustes.

Além dos casos de uso no aplicativo, este projeto também possui uma versão web. Esta versão foi pensada em ser uma alternativa ao uso do aplicativo, podendo ser usado nas televisões da enfermaria, na tela de dashboard ilustrada na Figura 24, ou como site da aplicação, possuindo todas as funcionalidades da aplicação no caso de uso médico. Nele, ao clicar na cama, temos acesso ao gráfico com os parâmetros monitorados, Figura 25, e na mesma tela, o histórico de alertas e sintomas daquela cama. Esta versão também possui o alerta de estado clínico do paciente, conforme a Figura 26.

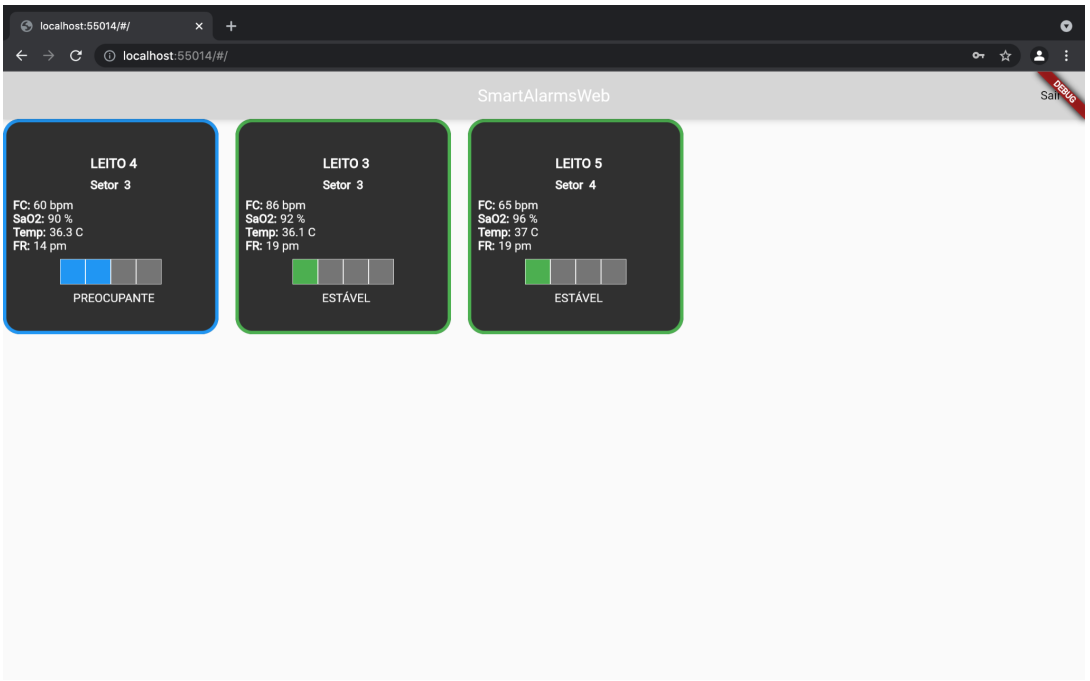

Figura 24. Tela de Dashboard Web, que pode ser usada em televisões na enfermaria.

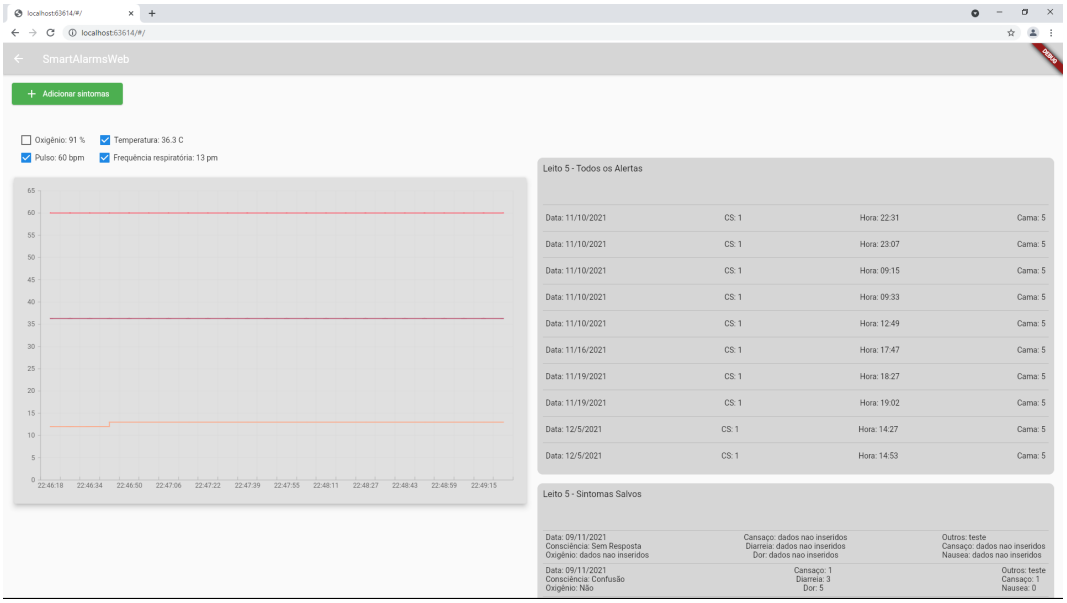

Figura 25. Tela de Detalhes Web.

Figura 26. Detalhe de alerta dos leitos.

6.2 Extensão das funcionalidades do backend

Conforme mencionado anteriormente, as mensagens do Protocolo MQTT são separadas por tópicos, que possuem seus níveis separados por barras, conforme mostrado na Figura 27. Por exemplo, o TOPIC_200 é o tópico relacionado aos dados dos setores, porém, para que a aplicação comece a receber os dados de um setor específico, é necessário que ocorra uma complementação ao texto do TOPIC_200, como visto na Figura 28. O complemento ('/#') no texto do tópico, tem como objetivo de explicitar que se quer receber tudo que estiver abaixo daquele nível.

```
/* ================================================
CONSTANTES DOS TOPICOS
================================================ */
String TOPIC_200 = "SmartAlarm/Data/";
String TOPIC_301 = "SmartAlarm/Alarms/Issued/";
String TOPIC_302 = "SmartAlarm/Alarms/Recognized/";
String TOPIC_303 = "SmartAlarm/Alarms/Cancelled/";
String TOPIC_401 = "SmartAlarm/Server/Application/Login/";
String TOPIC_402 = "SmartAlarm/Server/Application/InitialData/";
String TOPIC_601 = "SmartAlarm/Client/Application/Login/";
String TOPIC_602 = "SmartAlarm/Client/Application/InitialData/";
String TOPIC_604 = "SmartAlarm/Client/Application/History/";
String TOPIC_605 = "SmartAlarm/Client/Application/Query";
String TOPIC_405 = "SmartAlarm/Server/Application/Query";

var clientInitialData = TOPIC_602 + appId;
var serverInitialData = TOPIC_402 + appId;
var clientLoginTopic = TOPIC_601 + appId;
var serverLoginTopic = TOPIC_401 + appId;
```

Figura 27. Exemplos de tópicos utilizados no projeto.

```
var sectorData = TOPIC_200 + sectorId + '/#';
```

Figura 28. Exemplos de tópicos utilizados no projeto.

Na implementação deste projeto, tratamos o caso do paciente monitorado fora do hospital como se ele estivesse em outro setor somente com a sua cama, porém, seria totalmente possível fazer um tópico em que fossem enviadas mensagens do estado de um único sensor.

```
area/ID_da_area/sensor/ID_do_sensor/temperatura
area/ID_da_area/sensor/ID_do_sensor/umidade
```

```
area/10/sensor/5000/temperatura
area/10/sensor/5000/umidade
area/10/sensor/5001/temperatura
area/10/sensor/5001/umidade
area/20/sensor/4000/temperatura
area/20/sensor/4000/umidade
area/20/sensor/4001/temperatura
area/20/sensor/4001/umidade
```

Figura 29. Outra forma de uso dos tópicos [15].

Na implementação acima, Figura 29, retirado deste artigo sobre protocolos MQTT [15] , o tópico recebe, de forma similar a um parâmetro de função, o id do sensor no qual se deseja receber as mensagens. Assim, caso a aplicação precise somente dos dados do sensor 5000, basta apenas se subscrever nos tópicos de temperatura e umidade passando o 5000 como id do sensor na construção da string do tópico.

As conexões feitas do cliente ao broker possuem níveis de Qualidade de Serviço (QoS). Estes níveis podem ser:

1. QoS0 ou *at most once*: Semelhante ao protocolo UDP pois não se tem confirmação do recebimento da mensagem. Sendo assim, não se tem a obrigação de armazenar a mensagem para uma futura retransmissão.
2. QoS1 ou *at least once*: Existe a confirmação do recebimento de pelo menos uma mensagem. Este nível pode ser utilizado quando várias mensagens iguais são enviadas e quer se garantir que pelo menos uma delas terá resposta. A mensagem é armazenada até que se tenha a confirmação de recebimento.
3. Qos2 ou *exactly once*: Existe a confirmação do recebimento da mensagem nos dois sentidos, ou seja, o cliente envia uma mensagem e o broker que confirma o recebimento da mesma. Quando o cliente recebe a confirmação, ele confirma que recebeu a mensagem do broker. Assim, se

é garantido que a mensagem será enviada e recebida exatamente uma vez.

A escolha de qual QoS utilizar irá depender do contexto da aplicação. Sendo assim, não existe um QoS melhor ou pior, mas sim um que se adeque melhor ao caso de uso. A Figura 30 mostra um exemplo de subscribe utilizado no aplicativo. Como este subscribe é relacionado com o recebimento dos dados do setores, seria importante garantir que a mensagem seria recebida pelo menos uma vez pelo broker. Neste caso, também poderia ser possível utilizar o QoS exactly once, uma vez que ele também possui a confirmação do recebimento da mensagem.

```
_client.subscribe(sectorData,MqttQos.atLeastOnce);
```

Figura 30. Exemplos subscribe em um tópico.

Além dos tópicos de recebimento de mensagens dos dados dos sensores, possuímos também tópicos relacionados a login, recebimento e reconhecimento de alarmes dos leitos monitorados e consultas no banco de dados.

No *backend*, possuímos o PostgreSQL como banco de dados relacional para o armazenamento de dados utilizados na aplicação. A Figura 34 é um exemplo de como está sendo feito o envio de mensagens relacionadas ao banco de dados. Primeiro, possuímos duas variáveis de tópicos: ***bdClient*** e ***dbServer.*** A primeira delas está relacionada ao tópico do cliente, no qual iremos nos subscrever e também por onde receberemos o ***payload*** com a resposta da consulta (*query*) feita. Já a segunda variável está relacionada ao tópico do servidor, no qual enviaremos a mensagem.

```
void makePGQuery(queryName, bedId) {
  print("[Mqtt Web]: Connect to postgres");

  queryHolder = queryName;

  var bdClient = TOPIC_605 + '/$userId';
  var bdServer = TOPIC_405 + '/$userId';

  _client.subscribe(bdClient, MqttQos.atLeastOnce);

  Map<String, dynamic> str = {
    'QN': '$queryName',
    'UN':
        '${Provider.of<BedProvider>(contextNavigation, listen: false).currentUserName}',
    'BN': '$bedId',
  };

  String json = jsonEncode(str);
  Uint8List data = utf8.encode(json);
  Uint8Buffer dataBuffer = Uint8Buffer();
  dataBuffer.addAll(data);

  _client.publishMessage(bdServer, MqttQos.atLeastOnce, dataBuffer);
}
```

Figura 31. Exemplos de query no banco de dados.

Em seguida, montamos a mensagem, Figura 31, contendo as seguintes características:

1. QN (query name): nome da query que precisa ser feita.
2. UN (username): usuário logado.
3. BN (bed number): número da cama.

Uma vez tendo a mensagem montada, transformamos ela em json e logo em seguida em um tipo que seja aceito pelo mqtt no Flutter e, somente após isso, podemos enviá-la ao ***dbServer.***

No backend, existe uma função chamada *on_message_ctrl,* que é chamada a cada recebimento de mensagem. Nela, é checado para qual tópico a mensagem é destinada e então é feito o tratamento dos dados recebidos, conforme mostrado na Figura 32.

```
    elif topics.TOPIC_405 in msg.topic:
        print("------- TOPICO 405------- CONTROLLER_MQTT -------")
        recv_query_req(client, msg.topic, msg.payload, DB_CONN)
```

Figura 32. Função chamada ao receber mensagens para o banco de dados.

Uma vez feito o redirecionamento para as funções adequadas, é feito a checagem do QN, para saber qual query será feita no banco, de acordo com a Figura 33. Possuímos 6 tipos de query name, sendo eles:

1. *allAlarms*: realiza a query que recebe todos os alarmes de todos os setores da aplicação.
2. *SymptomsByUser*: realiza a query que recebe dados dos sintomas do usuário logado.
3. *SymptomsByBed*: realiza a query que recebe dados dos sintomas da cama.
4. *AlarmsByBed*: realiza a query que recebe todos os alarmes de determinada cama.
5. *insertSymptoms*: realiza a query de inserção de dados de sintomas de determinada cama ou usuário.
6. *insertAlarm*: realiza a query de inserção de dados de alarmes recebidos de determinada cama ou usuário.

```python
def recv_query_req(client, topic, payload, DB_CONN):
    curr = DB_CONN.cursor()
    query = dict(json.loads(payload))
    queryChamada = query['QN']

    if queryChamada == 'allAlarms':
        response = get_all_Alarms(curr)
    elif queryChamada == 'SymptomsByUser':
        response = get_symptoms_by_user(curr, query['UN'])
    elif queryChamada == 'SymptomsByBed':
        response = get_symptoms_by_bed(curr, query['BN'])
    elif queryChamada == 'AlarmsByBed':
        response = get_alarms_by_bed(curr, query['BN'])
    elif queryChamada == 'insertSymptoms':
        response = insert_symptoms(DB_CONN, curr, query)
    elif queryChamada == 'insertAlarm':
        response = insert_alarm(DB_CONN, curr, query)

    curr.close()
    resp = json.dumps(response)
    send_query_resp(client, topic, resp)
```

Figura 33. Redirecionamento das queries do banco de dados.

Dependendo da query que é feita, são necessários diferentes parâmetros, que são recebidos na mensagem que foi enviada.

```
def get_symptoms_by_user(curr, username):
    #
    # Returns all symptoms from user
    #
    clause = """
                SELECT *
                FROM "SmartAlarmMobile_symptoms"
                WHERE userlogged = %s
            """
    curr.execute(clause, (username,))
    row = curr.fetchall()
    return row
```

Figura 34. Exemplo de query de sintomas do usuário.

Após a execução da query, Figura 34, é retornada a resposta da mesma, podendo ela conter informação ou não. Então, o recv_query_req chama a função que irá mandar a mensagem para o aplicativo, mandando a resposta da query no payload da mensagem mqtt, conforme mostrado na Figura 35.

```
def send_query_resp(client, topic, payload):
    topic = topic.replace("Server", "Client")
    print("topico -", topic)

    if paho.mqtt.client.Client == type(client):
        client.publish(topic, payload)
    else:
        send_kafka_message(client, topic, "new", payload)
```

Figura 35. Função do envio de resposta via MQTT.

Voltando a aplicação mobile, temos uma função chamada *onMessageArrived()*, que funciona de forma similar a *on_message_ctrl,* ou seja, ela é chamada toda vez que o aplicativo recebe uma nova mensagem MQTT. Nela, de forma similar ao visto acima, é feito o redirecionamento das mensagens de acordo com o tópico recebido fazendo com que cada tópico tenha seu tratamento específico. No caso das respostas do banco de dados, é chamada a mensagem *getPGQuery(),* de acordo com a Figura 36.

```
} else if (c[0].topic.contains(TOPIC_605)) {
  print("entrou topico 405 - ${c[0].topic} - ");
  getPGQuery(contentPayload);
}
```

Figura 36. Função chamada ao receber chamadas do banco.

Dentro dela, fazemos a distinção entre os tipos de queries e então montamos o objeto relacionado ao dado recebido e exibimos na aplicação, como mostra a Figura 37. Somente após a conversão dos dados em um objeto que é feito o *unsubscribe* no tópico do ***bdCliente***, ou seja, a aplicação se desinscreve do tópico pois não é necessário estar inscrito nele uma vez que a resposta da query feita já foi retornada.

```
void getPGQuery(contentPayload) {
  var bdCliente = TOPIC_605 + '/$userId';
  var content = jsonDecode(contentPayload);

  if (queryHolder == 'AlarmsByBed' || queryHolder == 'allAlarms' ) {

    List<Alert> res = List<Alert>();
    var aux;
    content.forEach((element) => {
          aux = Alert(element[0], element[1], element[2], element[3],
              element[4], element[5], element[6]),
          res.add(aux)
        });
    BedProvider bedProvider = Provider.of<BedProvider>(contextProvider, listen: false);
    bedProvider.setAlertsListByBed(res);

  } else if (queryHolder == 'SymptomsByBed') {

    List<Symptom> res = List<Symptom>();
    var aux;
    content.forEach((element) => {
          aux = Symptom(
              element[3],
              element[9],
              element[8],
              element[5],
              element[1],
              element[6],
              element[4],
              element[7],
              element[0],
              element[2],
              element[10]),
          res.add(aux)
        });

    BedProvider bedProvider = Provider.of<BedProvider>(contextProvider, listen: false);
    bedProvider.setSymptomListByBed(res);

  }
  _client.unsubscribe(bdCliente);
}
```

Figura 37. Transformação do payload da mensagem MQTT em objeto da aplicação.

```
void insertSymptomsQuery(
    queryName,
    String conscience,
    String diarrea,
    String date,
    String headache,
    String hour,
    String nausea,
    String others,
    String ox,
    String pain,
    String tiredness,
    String userlogged,
    String bednumber) {
  print("[Mqtt Web]: Connect to postgres - insert on db");

  var bdCliente = TOPIC_605 + '/$userId';
  var bdServer = TOPIC_405 + '/$userId';

  _client.subscribe(bdCliente, MqttQos.atLeastOnce);

  Map<String, dynamic> str = {
    'QR': '$queryName',
    'conscience': '$conscience',
    'diarrea': '$diarrea',
    'date': '$date',
    'headache': '$headache',
    'hour': '$hour',
    'nausea': '$nausea',
    'others': '$others',
    'ox': '$ox',
    'pain': '$pain',
    'tiredness': '$tiredness',
    'userlogged': '$userlogged',
    'bednumber': '$bednumber',
  };

  String json = jsonEncode(str);
  Uint8List data = utf8.encode(json);
  Uint8Buffer dataBuffer = Uint8Buffer();
  dataBuffer.addAll(data);

  _client.publishMessage(bdServer, MqttQos.atLeastOnce, dataBuffer);
}
```

Figura 38. Exemplo com query de inserção no banco.

Caso a query feita seja de inserção de dados no banco, exemplificado na Figura 38, o fluxo é bastante similar, mudando somente a primeira função

chamada no aplicativo. Nela, passamos os dados a serem inseridos na mensagem que será mandada para o backend. Usando como exemplo a inserção de sintomas do paciente, a mensagem irá possuir: QN (query name), estado de consciência do paciente, níveis de desconforto de sintomas não monitorados como diarréia, dor de cabeça, náusea, dor, cansaço, outros sintomas, número da cama, usuário logado e data.

O uso do banco de dados via MQTT faz com que o projeto fique em conformidade com a arquitetura do SH-SENS e, além disso, é bastante confiável e escalável, uma vez que caso seja feita a troca de banco no *backend*, o *frontend* não será afetado pois as chamadas para recebimento de dados não são realizadas nele. Sendo assim, será preciso fazer alterações em somente em um lugar do projeto, evitando retrabalho.

7 Conclusão

O objetivo deste trabalho é dar continuidade ao Projeto do Sistema Inteligente de Monitoramento Paramétrico de Pacientes em enfermarias – COVID-19 (SH-Sens) [5]. Minha proposta era implementar um interface front end, que ficará na camada de aplicação do framework proposto no SH-Sens. A aplicação mobile se comunica com as demais camadas por meio de APIs que foram estendidas e adaptadas para suportar todos os casos de uso implementados.

Acredito que os objetivos principais do trabalho foram alcançados. Foi gasto um tempo considerável para entender o funcionamento do MQTT, seu funcionamento com o framework Flutter, sua integração como o backend do SH-SENS e a sua adaptação para o site web, visto que a biblioteca mqtt no aplicativo e no site possuem algumas diferenças significativas. Além disso, houve uma preocupação em respeitar o guia de estilo de cada sistema operacional, testes unitários foram realizados para garantir o funcionamento de algumas funções críticas do aplicativo, há uma preocupação com a Lei Geral de Proteção de Dados uma vez que há transparência no uso de dados do usuário e fazemos o acompanhamento de dados clínicos não monitorados pelos sensores, para ter uma visão mais completa do estado do paciente.

Apesar dos objetivos terem sido alcançados, acredito que alguns pontos podem ser implementados visando a melhoria do projeto. Um destes pontos seria a inserção da pressão arterial nos dados monitorados pelos sensores, para ter a informação completa do paciente monitorado. Um segundo ponto seria, na tela de detalhes do paciente, onde é exibido o gráfico com suas informações ao

longo do tempo, ter um gráfico mais customizado de acordo com o caso paciente, pois as vezes pode ser mais útil ver a variação de seus dados ao longo de horas e não ao longo dos minutos.

Por último, poderia ser implementado, junto com o protocolo NEWS 2, a escala Glasgow, que possui o objetivo de avaliar o nível de consciência dos pacientes e, dependendo de qual pontuação o paciente tenha, é possível determinar qual tipo de ação será preciso realizar.

8 Referências Bibliográficas

[1] ESTADO DO RIO REGISTRA MENOR TAXA DE OCUPAÇÃO DE LEITOS PARA COVID-19 DESDE O INÍCIO DA PANDEMIA .Disponível em https://coronavirus.saude.rj.gov.br/estado-do-rio-registra-menor-taxa-de-ocupacao-de-leitos-para-covid-19-desde-o-inicio-da-pandemia/. Acesso em: 18 novembro. 2021.

[2] Em 1 ano de pandemia, tecnologia se torna central para a vida do brasileiro.Disponível em https://www.poder360.com.br/coronavirus/em-1-ano-de-pandemia-tecnologia-se-torna-central-para-a-vida-do-brasileiro/. Acesso em: 18 novembro. 2021.

[3] Venda de celular sobe 10% no Brasil; auxílio emergencial ajudou, diz estudo. Disponível em https://www.uol.com.br/tilt/noticias/redacao/2021/01/22/venda-de-celular-sobe-no-3-tri-de-2020-no-pais-auxilio-emergencial-ajudou.htm. Acesso em: 18 novembro. 2021.

[4] Brasil registrou mais de 234 milhões de acessos móveis em 2020. Disponível em:https://www.gov.br/pt-br/noticias/transito-e-transportes/2021/05/brasil-registrou-mais-de-234-milhoes-de-acessos-moveis-em-2020#:~:text=Em%20dezembro%20de%202020%2C%20o,a%20tecnologia%203G%20e%204G. Acesso em: 18 novembro. 2021.

[5] PAGANELLI, Antonio Lyda et al. Projeto do Sistema Inteligente de Monitoramento Paramétrico de Pacientes em enfermarias – COVID-19. Disponível em https://docs.google.com/document/d/1tcdr2bNmO8MPpwiS5sbWAgeZhVqZHcmAg6Y21GOpdbE. Acesso em: 8 abril. 2021.

[6] CASAS INTELIGENTES: A REVOLUÇÃO DA INTERNET DAS COISAS. Disponível em https://concepthouse.com.br/casas-inteligentes-a-revolucao-da-internet-das-coisas/. Acesso em: 29 dezembro. 2021.

[7] Vias navegáveis limpas com inteligência artificial e IoT. Disponível em https://new.siemens.com/br/pt/empresa/stories/industria/yorkshirewater-agua-ai-iot-uk.html. Acesso em: 29 dezembro. 2021.

[8] IoT na saúde: tecnologia e evolução nos hospitais. Disponível em https://www.biocam.com.br/iot-na-saude-tecnologia-e-evolucao-nos-hospitais/. Acesso em: 05 abril. 2021.

[9] PHELAN, David. iPhone & Apple Watch Could Assess Heart Condition, Stanford Study Finds. Disponível em https://www.forbes.com/sites/davidphelan/2021/03/28/iphone--apple-watch-could-assess-heart-condition-stanford-study-finds/. Acesso em: 05 abril. 2021.

[10] LIMA, Ana Luiza. Teste de caminhada 6 minutos: o que é, para que serve e como fazer. Disponível em https://www.tuasaude.com/teste-de-caminhada-de-6-minutos/. Acesso em: 05 abril. 2021.

[11] Por que investir no monitoramento remoto de pacientes?. Disponível em https://previva.com.br/monitoramento-remoto-de-pacientes/. Acesso em: 05 abril. 2021.

[12] PRUTSACHAINIMMIT, J. T. et al. Development of Inpatient Monitoring System: A Case Study of Patong Hospital, Phuket, Thailand. Disponível em https://ieeexplore.ieee.org/document/9158062. Acesso em: 30 março. 2021.

[13] DUARTE, Rafael. NEWS2: Nova atualização para avaliação do risco de deterioração em diferentes cenários médicos. Disponível em https://pebmed.com.br/news2-nova-atualizacao-para-avaliacao-do-risco-de-deterioracao-em-diferentes-cenarios-medicos//. Acesso em: 18 novembro. 2021.

[14] MQTT. Disponível em https://pt.wikipedia.org/wiki/MQTT. Acesso em: 24 dezembro. 2021.

[15] MQTT - Protocolos para IOT. Disponível em https://www.embarcados.com.br/mqtt-protocolos-para-iot/. Acesso em: 24

dezembro. 2021.

[16] Flutter Dev. Disponível em https://flutter.dev/. Acesso em: 12 junho 2021.

[17] Arquitetura Flutter. Disponível em https://flutter.dev/docs/resources/architectural-overview. Acesso em: 23 junho de 2021.

[18] painting library. Disponível em https://api.flutter.dev/flutter/painting/painting-library.html. Acesso em 23 junho de 2021.

[19] Material. Disponível em https://material.io/. Acesso em 23 junho de 2021.

[20] Human Interface Guidelines. Disponível em <https://developer.apple.com/design/human-interface-guidelines/>. Acesso em 23 junho de 2021.

[21]National Early Warning Score (NEWS) 2. Disponível em https://www.rcplondon.ac.uk/projects/outputs/national-early-warning-score-news-2. Acesso em 25 junho de 2021.

[22]. NEWS2: Nova atualização para avaliação do risco de deterioração em diferentes cenários médicos. Disponível em: https://pebmed.com.br/news2-nova-atualizacao-para-avaliacao-do-risco-de-deterioracao-em-diferentes-cenarios-medicos/. Acesso em 25 junho de 2021.

[23] National Early Warning Score 2 (NEWS2) on admission predicts severe disease and in-hospital mortality from Covid-19. Disponível em: https://sjtrem.biomedcentral.com/articles/10.1186/s13049-020-00764-3. Acesso em 26 junho de 2021.

[24] Iubenda. Disponível em: https://www.iubenda.com/pt-br/. Acesso em 26 junho de 2021.