CENTRO UNIVERSITÁRIO DA FUNDAÇÃO ASSIS GURGACZ
DANIEL ZAGO DIAS

DESENVOLVIMENTO DE SOFTWARE VOLTADO À GESTÃO DE MANUTENÇÃO

CASCAVEL
2023

CENTRO UNIVERSITÁRIO DA FUNDAÇÃO ASSIS GURGACZ
DANIEL ZAGO DIAS

DESENVOLVIMENTO DE SOFTWARE VOLTADO A GESTÃO DE MANUTENÇÃO

Projeto de pesquisa apresentado ao Curso de Graduação em Engenharia Elétrica do Centro Universitário da Fundação Assis Gurgacz para elaboração do Trabalho de Conclusão de Curso II.

Orientador: Prof. Me. Elenilton Jairo Dezengrini.

CASCAVEL
2023

AGRADECIMENTOS

Gostaria de expressar meus sinceros agradecimentos a minha família e amigos, com um agradecimento especial a minha namorada, Pilar, e a minha mãe, Elisabete. A constante ajuda de minha namorada, tanto emocional quanto intelectual, desempenhou um papel crucial durante todo o processo de criação deste trabalho. Além disso, desejo expressar minha gratidão a todos os professores que dedicaram seu tempo para fornecer orientação e correções valiosas ao meu trabalho, em especial o meu orientador prof. Me. Elenilton Jairo Dezengrini.

RESUMO

O presente trabalho aborda os diferentes tipos de manutenção utilizados em organizações, com foco nas manutenções preventiva e corretiva. O principal objetivo da manutenção preventiva é manter a operação contínua de um sistema, evitando falhas inesperadas. Por outro lado, a manutenção corretiva é realizada após a ocorrência de falhas, com vistas a corrigir problemas nos equipamentos. A gestão de manutenção desempenha um papel crucial na organização e implementação de estratégias de manutenção eficazes, visando a otimização de custos e a melhoria da confiabilidade dos equipamentos. Além disso, a disponibilidade de sistemas é um fator essencial no contexto atual, com a proliferação de dispositivos móveis e a necessidade de desenvolvimento de aplicativos para diferentes plataformas. O uso do Flutter e do Dart como *framework* e linguagem de programação oferece uma solução eficiente e flexível para o desenvolvimento de aplicativos multiplataforma. A integração de dependências ajuda os desenvolvedores a aproveitar bibliotecas pré-desenvolvidas e simplificar o processo de desenvolvimento de aplicativos. A compreensão dos *widgets* e sua utilização é fundamental para a criação de interfaces de usuário personalizáveis e responsivas. Além disso, as dependências desempenham um papel crucial na expansão das funcionalidades do Flutter, permitindo o acesso a recursos e funcionalidades específicas. O trabalho apresenta um estudo de caso hipotético de uma indústria de fabricação de painéis fotovoltaicos no Brasil para ilustrar a aplicação dos conceitos discutidos. O aplicativo foi projetado com o intuito de atender essas necessidades. O desenvolvimento é detalhado, abrangendo a estrutura das telas e a lógica de programação. Um banco de dados é utilizado para armazenar informações dos usuários. O TCC destaca a importância da inovação e da tecnologia na gestão de manutenção, evidenciando o potencial contínuo de aprimoramento do aplicativo.

PALAVRAS CHAVES: Manutenção Corretiva e Preventiva. Gestão de Manutenção. Flutter & Dart.

ABSTRACT

This thesis will be exploring the different types of maintenance used within organizations, with focus on preventive and corrective maintenance. Preventive maintenance's primary objective is to maintain uninterrupted operation, preventing unexpected failures. In contrast, corrective maintenance is performed after na equipment failures occur, to rectify these issues. Maintenance management have a crucial role in establishing and implementing effective maintenances strategies, seeking to optimize costs and enhance equipment reliability. Additionally, in the current context marked by the growth of mobile devices and the need for cross-platform application development, system availability is essential. The use of Flutter and Dart as a framework and programming language provides an efficient and flexible solution for multi-platform app development. The use of dependencies assists developers in leveraging pre-developed libraries and simplifying the application development process. Understanding widgets and their utilization is fundamental in crafting customizable and responsive user interfaces. Furthermore, dependencies have a major role in expanding Flutter's capabilities, enabling access to specific features and functionalities. To illustrate the discussed concepts, a hypothetical case study of a photovoltaic panel manufacturing industry in Brazil is presented. The application was designed to meet the industry's needs, and the development process is detailed, encompassing screen structures and programming logic. A database is employed for user information storage. This thesis underscores the significance of innovation and technology in maintenance management, emphasizing the continuous potential for application enhancement.

KEYWORDS: Corrective and Preventive Maintenance. Maintenance Management. Flutter & Dart.

LISTA DE FIGURAS

Figura 1 – MTTR Mensal.....7
Figura 2 – Fluxograma Scrum.....14

LISTA DE ABREVIATURAS E SIGLAS

AOT	(*Ahead-Of-Time*) Tipo de compilação
JIT	(*Just-In-Time*) Tipo de compilação
ART	Anotação de Responsabilidade Técnica
MTTR	(*Mean Time to Repair*) Tempo Médio para Reparar
GPS	(*Global Positioning System*) Sistema de posicionamento global
NFC	(*Near Field Communication*) Comunicação por campo de proximidade
APP	(*Application*) Aplicativo
WEB	(*World Wide Web*) Rede mundial de computadores
IDE	(*Integrated Development Environment*) Ambiente de desenvolvimento integrado
DS	(*Daily Scrum*) Reunião diária na qual se discute a evolução do projeto
PNG	(*Portable Network Graphic*) Gráficos portáteis de rede
GIF	(*Graphics Interchange Format*) Formato de Intercâmbio Gráfico
JPEG	(*Joint Photographic Experts Group*) Grupo responsável pela criação do formato
URL	(*Uniform Resource Locator*) Forma padronizada de representação de diferentes documentos, mídia e serviços de rede na internet
WEBP	Formato de arquivo que foi desenvolvido pela Google
BMP	(*Device Independent Bitmap*) Formato de gráficos por mapa de bits

SUMÁRIO

1 INTRODUÇÃO 1

2 FUNDAMENTAÇÃO TEÓRICA 3

2.1 Tipos de manutenções 3

2.2 Gestão de manutenção 5

2.3 Disponibilidade de sistemas 6

2.4 Indicadores gráficos 7

2.5 Desenvolvimento de aplicativos 7

2.6 Flutter & Dart 8

2.6.1 *Widgets* 10

2.6.2 *Dependencies* 11

2.7 Metodologia Scrum 13

3 MATERIAIS E MÉTODOS 14

3.1 Ferramentas Utilizadas 14

3.2 *Softwares* Utilizados 15

4 DESENVOLVIMENTO 16

4.1 Estrutura de telas 16

4.1.1 Tela de cadastro e login 16

4.1.2 Tela setores 16

4.1.3 Tela de máquinas 16

4.1.4 Tela de adicionar usuários 18

4.1.5 Tela de indicadores 18

4.2 Banco de dados 18

4.3 Testes e Ajustes 18

5 TRABALHOS FUTUROS 19

5.1 Aperfeiçoamento da visualização de Informações 19
5.1.1 Aprimoramento da tela de Indicadores 19
5.1.2 Seleção de fichas 19
5.1.3 Visualização de apontamentos 19
5.2 Melhorias de *design* 20
5.2.1 Atualização de ícones 20
5.2.2 Revisão do *layout* 20
5.2.3 Aprimoramento de *dialogs* 20
5.2.4 Paleta de cores 20
6 CONSIDERAÇÕES FINAIS 21
REFERÊNCIAS BIBLIOGRÁFICA 22
APÊNDICES 26

1 INTRODUÇÃO

O acesso à informação sobre máquinas em uma empresa é importante. Entretanto, ainda são utilizados métodos ineficientes para realizar esse registro, além de haver possibilidade dessa informação ser perdida ou até mesmo não registrada corretamente. Isto pode influenciar na análise tempo de máquina parada.

Tempo de máquina parada faz com que se tenha uma grande perda de recurso monetário para a empresa, em que esta pode não produzir o suficiente, em virtude da parada. A fim de reduzir tal efeito, é necessário utilizar-se de métodos de gestão de manutenção.

Muitas empresas ainda não fazem a aplicação correta da gestão de manutenção. De acordo com Fernandes *et al*. (2015), as principais dificuldades de implementação de um sistema de gestão de manutenção eficiente são a complexidade dos processos de manutenção, a falta de padronização de dados, a falta de apoio da administração e a falta de treinamento adequado dos funcionários.

A aplicação de gestão de manutenção é necessária para manter um bom funcionamento dos equipamentos de uma empresa. Além disso, ela ajuda a manter competitividade em um mercado em que a eficiência é a chave.

A manutenção é um processo essencial para garantir o funcionamento nas empresas industriais. Tendo como objetivo garantir a disponibilidade e a confiabilidade dos equipamentos, a gestão de manutenção requer organização e planejamento. Diante desse cenário, a utilização de *softwares* de gestão de manutenção tem se mostrado uma alternativa eficiente, para melhorar a gestão e a tomada de decisão nesse processo (FREITAS, 2018).

A gestão de manutenção em uma empresa industrial pode ser uma tarefa complexa e desafiadora, especialmente em relação à organização e ao planejamento das atividades. Por isso, já existem diversos métodos para se manter uma boa gestão de manutenção. Para que se possam ter várias vantagens, são efetuadas ações como a redução dos custos de produção, evitando com que as máquinas apresentem defeitos inesperados em momentos de produção e, por conseguinte, também aumentando a disponibilidade e confiabilidade dos equipamentos, de sorte a garantir a qualidade e a segurança da empresa (OTANI; MACHADO, 2018).

Segundo Yamakawa, Miguel e Aoki (2014), um *software* de gestão pode melhorar a eficiência e a eficácia da manutenção, ajudando a identificar áreas de melhoria e reduzir o tempo de inatividade não programado.

Neste contexto, propõe-se a utilização de linguagem de programação, com vistas a criar um aplicativo de gestão de manutenção corretiva e preventiva, mediante a adoção de um

método, para realizar o seu registro e análise.

Busca-se desenvolver um *software* capaz de registrar fichas, preventivas e manutenções corretivas de máquinas industriais, com o objetivo de gerar dados, para análise da manutenção, além de dados importantes voltados à gestão de manutenção. Utiliza-se uma interface intuitiva e simplificada, para ser de fácil utilização.

Além de facilitar a comunicação dessas manutenções, um *software* que permita a visualização e o registro de manutenções feitas, com previsão do acesso pela equipe responsável pela gestão de manutenção, possibilita o manuseio dos dados gerados na realização de análises de equipamento que mais geram defeitos, ademais de proporcionar a quantificação do tempo de parada de cada máquina.

Já existem *softwares* que realizam a gestão de manutenção, como SAP® e Intelup®. Ambos têm como princípio o controle de defeitos através de funções automatizadas, tornando, assim, o processo de gestão mais simples e prático. O conceito dá-se em torno de uma disponibilidade de tecnologias englobadas aos equipamentos, o que limita a aplicação dessas ferramentas, além do seu valor de compra, que pode ser elevado, levando em consideração o mercado brasileiro.

A capacidade do Flutter de criar interfaces do usuário ricas e responsivas é significativa. Por meio de seus *widgets* personalizáveis e animações fluidas, o Flutter permite oferecer uma experiência visualmente e atraente aos usuários do aplicativo de gestão de manutenção. Além disso, o Flutter possui recursos para o desenvolvimento de interfaces adaptáveis a diferentes tamanhos de tela e orientações, proporcionando uma experiência consistente em dispositivos variados (FAUST, 2020).

Por fim, a escolha do Flutter também se deve ao seu desempenho e eficiência. O compilador possui uma forma de visualizar as alterações em tempo real, o que permite realizar alterações e correções em tempo real durante o desenvolvimento, agilizando o processo e aumentando a produtividade. A linguagem também utiliza compilação AOT (*Ahead of Time*) e JIT (*Just-in-Time*), o que resulta em um desempenho rápido e eficiente do aplicativo. A compilação AOT e JIT traduzem o código-fonte diretamente para a linguagem de máquina, reduzindo o tempo de inicialização e melhorando a velocidade de execução (OLIVEIRA; SILVA, 2013).

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Tipos de manutenções

Existem várias classificações para tipos diferentes de manutenção. Para quesito organizacional, abordar-se-ão dois tipos de manutenções diferentes neste trabalho, sendo a manutenção preventiva e a manutenção corretiva, que são as mais abrangentes possíveis.

2.1.1 Manutenção Preventiva

A manutenção preventiva consiste em monitorar e realizar visitas regulares para identificar sinais de possíveis falhas em equipamentos, com atividades como testes, substituições de peças e ajustes. Essa manutenção pode ser agendada com antecedência ou programada, de acordo com o tempo de operação dos equipamentos. Em alguns casos, regulamentos obrigam a realização da manutenção preventiva. Segundo Cabral (2009), a manutenção preventiva é feita em intervalos de tempo determinados ou de acordo com critérios estabelecidos, para reduzir a chance de falhas ou degradação do funcionamento de um bem, e pode ser classificada em manutenção sistemática e preventiva.

A manutenção preventiva tem como objetivo principal manter a operação contínua de um sistema, por meio de intervenções programadas em intervalos definidos. De acordo com a definição formal de Pinto e Xavier (2001), a manutenção preventiva é realizada para reduzir ou evitar a falha ou queda de desempenho do sistema. O princípio básico da manutenção preventiva é determinar o período de maior probabilidade de falha de um componente e o substituir antes que atinja o fim de sua vida útil, evitando falhas inesperadas no equipamento.

No entanto, a implementação da manutenção preventiva pode ser desafiadora, pois requer a identificação precisa do período em que as intervenções devem ser realizadas. Se o período é subestimado, os custos serão elevados devido à reposição desnecessária de componentes. Por outro lado, se o período é superestimado, há um risco maior de falha inesperada do sistema ou equipamento. Portanto, é essencial encontrar um equilíbrio entre esses dois extremos, para garantir a eficácia da manutenção preventiva (OTANI; MACHADO, 2008).

Para aumentar a confiabilidade na programação da manutenção preventiva, é necessário coletar uma grande quantidade de informações sobre os equipamentos a serem mantidos. Isto envolve cadastrar todos os equipamentos e concentrar informações, como suas características técnicas, localização, vida útil estimada, estoque de peças sobressalentes e contatos de empresas

de fornecimento e reparo. Essas informações serão usadas para criar fichas de equipamentos nas quais serão registrados os dados, como data da ocorrência, número de série, motivo da falha, material utilizado no reparo, tempo gasto, entre outros. Esse histórico do equipamento ajuda a aprimorar o processo de manutenção preventiva, pois permite registrar as características e o tempo de vida útil reais dos componentes do equipamento (SILVA, 2017).

As informações sobre os equipamentos elétricos ou mecânicos ajudam a definir a ordem de prioridades para a programação da manutenção preventiva. A implementação da manutenção preventiva oferece benefícios significativos em comparação com a manutenção corretiva, como a redução do estoque de sobressalentes, menor tempo de paradas não planejadas e aumento da confiabilidade dos equipamentos e instalações (FUENTES, 2006).

No entanto, a desvantagem da manutenção preventiva é a possibilidade de introduzir elementos, no sistema, que possam causar novas falhas, especialmente quando se adquire produtos com especificações inadequadas, levando a problemas antes inexistentes. Essa desvantagem pode ser controlada com medidas rigorosas de especificação, como a exigência de normas técnicas, laudos de ensaios com ART do profissional, termos de garantia e assistência técnica local, que ajudam a filtrar fornecedores indesejados (SILVA, 2017).

3.1.2 Manutenção Corretiva

Por outro lado, a manutenção corretiva, também chamada de curativa, é a forma mais básica de manutenção, realizada por meio de observação ou medição de equipamentos antes ou após falhas funcionais. Algumas organizações chamam-na de manutenção de reparação e é conduzida para corrigir erros e fazer o equipamento voltar a funcionar. Reparações importantes e reativas podem ser agendadas ou classificadas (CARVALHO, 2019).

A manutenção corretiva é uma ação de correção aplicada a um equipamento após a ocorrência de uma falha, com o objetivo exclusivo de solucionar problemas causados por avarias ou desgaste de um ou mais elementos do sistema ou equipamento. Segundo Silva (2017) a manutenção corretiva é conduzida quando o equipamento falha ou não atinge uma condição aceitável de operação ou desempenho. Essa manutenção pode ocorrer de forma não planejada, como resultado de falhas inesperadas, que surgem quando não há acompanhamento regular do estado operacional da instalação ou dos equipamentos, conforme confirmado por Pinto & Xavier (2001).

A manutenção corretiva é adequada para instalações e equipamentos de baixo custo de produção, em que a substituição ou a disponibilidade de reserva é menos custosa do que a

aplicação de outros métodos de manutenção. Nesses casos, é interessante intervir apenas quando ocorrer falência operacional do equipamento. Pinto & Xavier (2001) esclarecem que essa compreensão decorre do que é estabelecido sobre manutenção corretiva.

Nesse contexto, pode-se afirmar que a manutenção corretiva também deve ser aplicada em equipamentos com curto ciclo de vida devido à obsolescência tecnológica, como medidores de energia, relés digitais e medidores de temperatura digitais. Com o surgimento de novas tecnologias, torna-se necessária a rápida substituição desses equipamentos, tornando muitas vezes inviável uma manutenção planejada, uma vez que a obsolescência acontece antes mesmo do fim da vida útil do equipamento ou instalação (SILVA, 2017).

Assim, a detecção de falhas nesses componentes deve levar à substituição por equipamentos mais modernos, caracterizando uma manutenção corretiva em nível de substituição de instrumentos e não consertos pontuais.

No entanto, é importante destacar que esses casos devem ser considerados exceções. A utilização da manutenção corretiva como estratégia para todos os equipamentos pode causar problemas significativos, impactando, negativamente, no processo de trabalho ou na linha produtiva. Isso porque, ao optar por essa filosofia de manutenção, o sistema e seus componentes ficam sujeitos a falhas inesperadas e defeitos em cascata, resultando em um maior tempo de parada para manutenção, aumento expressivo dos custos de conserto, situações de insegurança para os profissionais e diminuição da operosidade e confiabilidade dos equipamentos (XENOS, 1998).

2.2 Gestão de manutenção

Para se compreender a gestão de manutenção, é necessário entender, primeiramente, o que é gestão. A definição de acordo Pombo (2011), em seu "Dicionário de Sinônimos da Língua Portuguesa", é administrar e superintender. Assim, a gestão de manutenção é o método de planejamento, organização, controle e execução de atividades de manutenção preventiva e corretiva, com o objetivo de minimizar falhas, aumentar a vida útil dos equipamentos, reduzir custos e evitar interrupções não programadas na produção.

Para manter uma boa disponibilidade de máquinas é necessário a implementação de métodos e ferramentas eficientes de gestão de manutenção. De acordo com Ramos (2012), a função da manutenção tem passado e ainda passa por uma evolução significativa, impulsionada pelo avanço tecnológico e científico. Como resultado, diversas estratégias de gestão foram desenvolvidas, com o intuito de promover uma maior segurança, reduzir o impacto ambiental

e melhorar a qualidade dos produtos ou serviços, sempre em busca da otimização dos custos.

Atualmente, muitas empresas estão cientes dos desafios enfrentados pela função de manutenção e adotam políticas e estratégias para elevá-la a mesma importância de outras funções organizacionais. Dessa forma, a função de manutenção torna-se uma parte integral das estratégias implementadas pela organização, visando se tornar a melhor no mercado (FUENTES, 2006). Além de reduzir os custos de operação, cada quantia investida em planejamento pode resultar em uma economia 5 (cinco) vezes maior em sua execução, 25 (vinte e cinco) vezes maior em manutenção preventiva e até 125 (cento e vinte e cinco) vezes maior em manutenção corretiva (ROSCOFF *et al.*, 2020).

2.3 Disponibilidade de sistemas

De acordo com a empresa TeleGeography® (2021), no mundo, é estimado que o número de dispositivos móveis conectados à internet está próximo a 1,1 por pessoa. No Brasil, de acordo com a Fundação Getúlio Vargas (FGV) (2020), existem 234 (Duzentos e trinta e quatro) milhões de aparelhos moveis para, 213 (Duzentos e treze) milhões de habitantes, assim tendo aproximadamente 1,1 celulares por pessoa.

Foi tendo em vista a vasta disponibilidade desses aparelhos que se deu a necessidade de *softwares* compatíveis com eles. Assim, a ferramenta desenvolvida no dispositivo em questão pode alcançar o máximo de beneficiários possível. De acordo com Stoll (2018), além da disponibilidade de ferramentas e sensores que há nos aparelhos *smart,* que podem incluir, mas não se limitar a GPS, giroscópio, câmeras, NFC, emissor infravermelho, acesso à internet, entre muitas outras disponíveis, que podem variar de modelos de aparelhos diferentes. Para o desenvolvimento do *software* orientado no trabalho, tende-se a utilizar apenas as funções mais comuns, como acesso à internet e funções de toque.

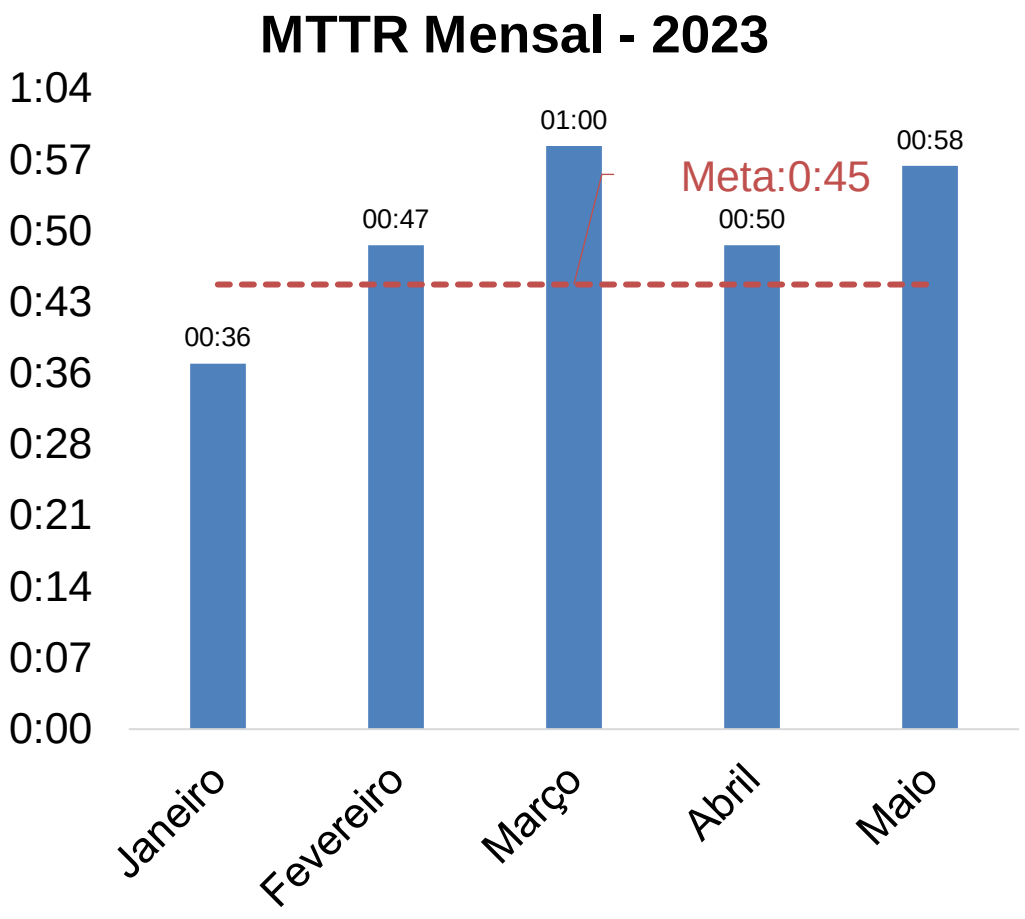
Levando em consideração as estatísticas apresentadas pela empresa Starcounter® (2023), em fevereiro de 2023, cerca de 72,26% dos aparelhos moveis do mundo utilizam sistema operacional Android®. Consequentemente, o APP que disponibiliza o *download* de outros APPs é o Google Play®. Tendo em vista essa informação, o aplicativo será desenvolvido para atender as normas da plataforma mencionada, para que seja possível a disponibilização.

2.4 Indicadores gráficos

MTTR, ou Tempo Médio para Reparo, é uma métrica utilizada na área de manutenção para medir o tempo médio necessário para reparar uma falha ou resolver um problema em um equipamento. O MTTR é calculado dividindo o tempo total de parada devido a uma falha ou problema pelo número de ocorrências (ROSA, 2006).

Pode ser observado um exemplo de gráfico de MTTR na imagem a seguir, Figura 1. Este é um dos principais indicadores referentes a manutenção:

Figura 1 – MTTR Mensal



Fonte: Autor

2.5 Desenvolvimento de aplicativos

Na programação, há o que se chamado de desenvolvimento nativo e programação multiplataforma (*cross-platform*). Desenvolvimento nativo é um processo em que um aplicativo

é criado especialmente para uma plataforma específica, como iOS® ou Android®. Isto significa que todas as funções da plataforma podem ser utilizadas sem limitações e existem padrões específicos para a aparência e experiência do usuário. Ou seja, quando se desenvolve um aplicativo nativo, ele é projetado especificamente para funcionar em um tipo de dispositivo e sistema operacional (PREZOTTO; BONIATI, 2014). Já em multiplataforma, esse tipo de desenvolvimento consiste em criar uma página WEB, usando tecnologias como HTML, CSS e *JavaScript*, que pode ser exibida dentro de um "*container*" no interior de um aplicativo nativo. Isto significa que apenas um código precisa ser criado para funcionar em várias plataformas diferentes, sem precisar de especialistas em cada plataforma. No entanto, nem todas as funcionalidades da plataforma estão disponíveis para serem utilizadas, como acontece no desenvolvimento nativo (MATOS; SILVA, 2016).

2.6 Flutter & Dart.

A Linguagem Dart foi desenvolvida em 2011, na Dinamarca, pelos desenvolvedores Lars Bark e Kasper Lund. A linguagem veio com o objetivo de programação *WEB* e é baseada na compilação de código *JavaScript.*

O Flutter é uma ferramenta de desenvolvimento com enfoque em várias plataformas para dispositivos móveis, que foi desenvolvida pela Google®. Ele tem sido amplamente empregado no mercado e, agora, tem a capacidade de ser utilizado para construir aplicações em sistemas operacionais como Linux®, Windows®, Android® e iOS® (FRANKLIN; FILHO, 2020).

O Flutter é uma ferramenta de programação que ajuda a criar aplicativos para *smartphones*. Ele usa pequenos blocos chamados "*widgets*" para construir botões, menus, campos de texto e outros elementos que são visualizados nos *smartphones*. O Flutter é diferente de outras ferramentas pois permite que os programadores criem seus próprios *widgets* combinando com outros *widgets* conjuntamente. Essa flexibilidade permite criar interfaces únicas e personalizadas para cada aplicativo. Outras ferramentas de programação para iOS® e Android® têm limitações na criação de interfaces personalizadas. Isso só e possível por utilizar uma estrutura em camadas. Essa abordagem permite que os desenvolvedores personalizem e criem interfaces exclusivas e complexas. A equipe do Flutter também criou os *widgets* de alto nível existentes, usando essa mesma técnica. Não há barreiras para que os desenvolvedores façam o mesmo, o que oferece uma flexibilidade de personalização em comparação com outras ferramentas de programação de interface do usuário, como iOS® e Android®, que têm

implementações hierárquicas e níveis de acesso limitados (DAGNE, 2019).

Para realizar a programação da linguagem é necessário utilizar um editor IDE de sua escolha. Este será responsável por compilar, simular o código, além de fornecer ferramentas para a facilitação da programação. A escolha do compilador não interfere n a criação do código e é possível programar até mesmo em um bloco de notas do Windows®, embora não ofereça nenhum recurso.

Com o Flutter, é possível organizar o código em pastas para torná-lo mais eficiente e fácil de gerenciar. A pasta principal é a "*lib*", onde todo o código-fonte do aplicativo é armazenado. Dentro dessa pasta, pode-se criar subpastas, para organizar ainda mais o código em diferentes áreas. Essa estrutura é importante para manter o código organizado e facilitar a manutenção e desenvolvimento do aplicativo.

A pasta "*model*" contém o modelo de objetos que será usado para armazenar informações em um banco de dados.

A pasta "*ui*" contém todo o código que cuida da aparência do aplicativo e da forma como os usuários interagem com ele.

A pasta "*util*" contém todo o código de utilidade e classes auxiliares para ajudar na criação do aplicativo.

Importante lembrar que essa é apenas uma maneira de organizar o código, podendo haver a possibilidade de se escolher organizar o código de outras maneiras. O Flutter não impõe nenhuma estrutura específica de pastas para o projeto. Porém, seguir essa estrutura pode facilitar o entendimento do processo de desenvolvimento do aplicativo (FREITAS, 2019).

Entender o funcionamento da linguagem Dart é fundamental. Os fundamentos do Dart são semelhantes aos de todas as linguagens de alto nível. Caso o desenvolvedor venha do JavaScript, Java ou qualquer outra linguagem com sintaxe semelhante à linguagem C, encontrará familiaridade na sintaxe do Dart. Caso o desenvolvedor do Ruby ou Python, sentir-se-á à vontade com o *design* orientado a objetos do Dart (WINDMILL, 2020).

O *framework* Flutter é a principal razão pela qual o Dart é amplamente utilizado no desenvolvimento de aplicativos móveis. O Flutter é um *framework* de código aberto que permite criar aplicativos móveis multiplataforma de alta qualidade para Android® e iOS®, a partir de um único código-fonte. O Dart é a linguagem de programação central usada para desenvolver a lógica do aplicativo, no Flutter (YOU; HU, 2021).

Uma das características mais marcantes do Flutter é sua arquitetura centrada em *widgets*. Os desenvolvedores podem criar interfaces de usuário personalizáveis e responsivas usando uma ampla variedade de *widgets* Flutter, muitos dos quais são escritos em Dart. Isto simplifica

a criação de interfaces de usuário bonitas e interativas para aplicativos móveis (FAUST, 2020).

Além disso, o Dart e o Flutter oferecem excelente suporte para a integração com recursos nativos em dispositivos móveis. Isto significa que os desenvolvedores podem acessar facilmente recursos como a câmera, sensores, GPS e notificações, por meio de pacotes e *plugins*, tornando mais fácil criar aplicativos ricos em recursos (STOLL, 2018).

2.6.1 *Widgets*

Um *widget* é uma unidade de interface de usuário que define um elemento específico na tela, como um botão, um campo de texto ou uma imagem. Cada *widget* é responsável por renderizar uma parte da interface do usuário e responder a eventos, como toques na tela (LOUGHEED, 2023).

De acordo com Morgan e Alessandro Biessek, em seu livro "*Flutter for beginners*" (p. 122-140), existem diversos tipos de *widgets*. Seguem vários exemplos:

Widgets de Texto: Alessandro Biessek afirma em seu livro "*Flutter for beginners*" (p. 122-140) que *widget Text* é usado para exibir texto a partir de uma *String* e possui capacidade para estilização por parte do desenvolvedor, sendo seus itens:

style: Esta propriedade é responsável por configurar a estilização do texto. A propriedade *style* proporciona controle total sobre o visual do texto, tornando possível personalizar cada aspecto da sua apresentação.

textAlign: O *textAlig* controla o alinhamento horizontal do texto, permitindo opções como centralizado e justificado, entre outras.

maxLines: Com a propriedade *maxLines*, é possível especificar um número máximo de linhas para o texto. Caso o conteúdo exceda esse limite, o texto será cortado, de acordo com a definição, proporcionando controle sobre o espaço ocupado.

overflow: A propriedade *overflow* determina como o texto será tratado em caso de overflow, ou seja, quando não cabe no espaço disponível. Ela oferece opções para especificar um limite máximo de linhas ou adicionar uma elipse no final do texto, garantindo que a exibição seja gerenciada de forma adequada.

*Widget*s de Imagem: O widget *Image* exibe imagens de diferentes fontes e formatos. De acordo com a documentação, os formatos de imagem suportados incluem JPEG, PNG, GIF, GIF animado, WEBP, WEBP animado, BMP e WBMP (BIESSEK, 2023, p. 122-140).

A propriedade *Image* do widget especifica um *ImageProvider*, para indicar a imagem a ser exibida, que pode ser proveniente de várias fontes. A classe *Image* oferece diferentes

construtores para carregar imagens de maneiras distintas. Por exemplo, o construtor Image.network (URL) utiliza *NetworkImage* para obter uma imagem a partir de uma URL.

Widgets de Botão: Os *widgets* de botão são interativos e respondem a eventos de toque. (BIESSEK, 2023, p. 122-140). Eles incluem *widgets* como:

RaisedButton: Um botão elevado *Material Design*. Um botão elevado consiste em uma peça retangular de material que paira sobre a interface.

FloatingActionButton: Um botão flutuante é um botão que paira sobre o conteúdo para promover uma ação principal na aplicação.

DropDownButton: Mostra o item atualmente selecionado e uma seta que abre um menu para selecionar outro item de uma lista determinada.

PopUpMenuButton: Exibe um menu quando pressionado.

Widgets de Entrada de Dados: O *widget TextField* permite que os usuários insiram texto. É personalizável e oferece a capacidade de definir restrições, como número máximo de caracteres e validação de entrada. *Widgets* como *Checkbox* e *RadioButton* são usados para coletar seleções de opções, como caixas de seleção e botões de opção (BIESSEK, 2023, p. 122-140).

2.6.2 *Dependencies*

No desenvolvimento de aplicativos móveis com Flutter, as dependências ou *dependencies* desempenham um papel fundamental. Elas são bibliotecas de código pré-desenvolvido, que oferecem funcionalidades específicas, ampliando as capacidades do Flutter e simplificando o desenvolvimento de aplicativos. As dependências permitem que os desenvolvedores acessem uma ampla gama de recursos e funcionalidades sem a necessidade de criar tudo do zero (AQEEL, 2021).

As dependências, amplamente acessíveis através do *website* https://pub.dev/packages, podem ser criadas tanto por usuários quanto por desenvolvedores do Google®. Elas passam por atualizações contínuas e são submetidas à verificação por parte de outros membros da comunidade. Esse ecossistema visa a criação de uma plataforma diversificada, que disponibiliza uma vasta gama de ferramentas, para aprimorar aplicativos desenvolvidos em Flutter. Devido à natureza colaborativa e em constante evolução do ambiente, não é possível atribuir precisamente autoria individual a essas dependências.

Seguem as dependências utilizadas para o desenvolvimento do aplicativo:

path_provider: O pacote *path_provider* é uma dependência do Flutter que fornece

utilitários para trabalhar com caminhos de arquivos e diretórios. Ele é usado para acessar e manipular arquivos e pastas no sistema de arquivos do dispositivo. Isto é útil para tarefas como salvar ou ler arquivos locais, armazenar dados em cache ou acessar diretórios específicos no dispositivo.

O *path_provider* permite que os desenvolvedores obtenham caminhos para diretórios especiais, como o diretório de documentos, o diretório de cache, o diretório de *downloads* e outros. Isto é essencial para aplicativos que precisam armazenar dados localmente, como aplicativos de gerenciamento de arquivos, aplicativos de galeria de fotos e muito mais.

shared_preferences: O pacote *shared_preferences* é uma dependência do Flutter que permite armazenar e recuperar dados simples de forma persistente em formato de pares chave-valor. Isto é útil para armazenar configurações de aplicativos, preferências do usuário e outras informações que precisam sobreviver a reinicializações do aplicativo.

O *shared_preferences* usa um mecanismo de armazenamento compartilhado no dispositivo para manter esses dados, tornando-os acessíveis mesmo após o aplicativo ser fechado e reiniciado. Isto é comum em aplicativos que desejam lembrar configurações personalizadas, como preferências de tema, autenticação ou preferências do usuário.

fl_chart: O pacote *fl_chart* é uma biblioteca de gráficos para o Flutter, que facilita a criação e exibição de gráficos interativos em aplicativos. Ele oferece suporte a vários tipos de gráficos, como gráficos de barras, gráficos de pizza, gráficos de linhas e muito mais. O *fl_chart* permite personalizar a aparência dos gráficos e os tornar interativos, permitindo que os usuários explorem dados de maneira visual.

Essa biblioteca é valiosa para aplicativos que desejam visualizar dados de maneira eficaz, como aplicativos de análise, painéis de controle e aplicativos que exibem estatísticas em tempo real.

datetime_picker_formfield: O pacote *datetime_picker_formfield* é uma dependência que simplifica a captura de datas e horários em formulários. Ela fornece campos de entrada de data e hora personalizáveis, que exibem um seletor de data e hora quando o usuário toca neles. Isto é útil para aplicativos que envolvem agendamento, reservas, registro de eventos e outros cenários em que a seleção de datas e horas é necessária.

O *datetime_picker_formfield* ajuda a evitar a complexidade de criar seus próprios seletos de datas e horas e oferece uma experiência de entrada de dados mais amigável para o usuário.

get: O pacote *get* é uma biblioteca de gerenciamento de estado e navegação para o Flutter. Ele fornece uma maneira eficiente de gerenciar o estado do aplicativo e navegar entre

telas. O *get* oferece um gerenciador de estado simples e eficaz que permite que os desenvolvedores compartilhem dados entre *widgets* de forma rápida e eficiente.

cupertino_icons: *cupertino_icons* é uma biblioteca que fornece ícones no estilo *Cupertino*, que é o estilo de design da Apple®. Ela inclui um conjunto de ícones, que podem ser usados em aplicativos Flutter que desejam aderir ao design visual do iOS® da Apple®. Esses ícones são, frequentemente, usados em aplicativos que visam oferecer uma experiência de usuário semelhante à dos dispositivos Apple®, como iPhones e iPads.

2.7 Metodologia Scrum

O Scrum é uma metodologia ágil amplamente utilizada como uma ferramenta organizacional no desenvolvimento de projetos. Ele se baseia em um *framework*, estrutura de trabalho, que enfatiza a colaboração, a flexibilidade e a entrega incremental de resultados. O Scrum utiliza uma abordagem, na qual as decisões são baseadas em observação e experiência, e não em suposições ou previsões.

Durante o projeto ocorre o *Sprint Planning*. Nessa reunião, a equipe de desenvolvimento define os objetivos da *Sprint* e seleciona os itens de trabalho que serão realizados durante o período. A equipe estima o esforço necessário para concluir cada item de trabalho e define como eles serão implementados. O *Sprint Planning* é uma oportunidade para todos os membros da equipe colaborarem e alinhar suas expectativas em relação ao que será realizado.

Ao longo dos *Sprints*, a equipe realiza as DS. As *Daily Scrums* são reuniões curtas diárias, de aproximadamente 15 minutos, nas quais os membros da equipe compartilham o que foi realizado no dia anterior, o que será realizado no dia atual e quaisquer obstáculos que estejam enfrentando.

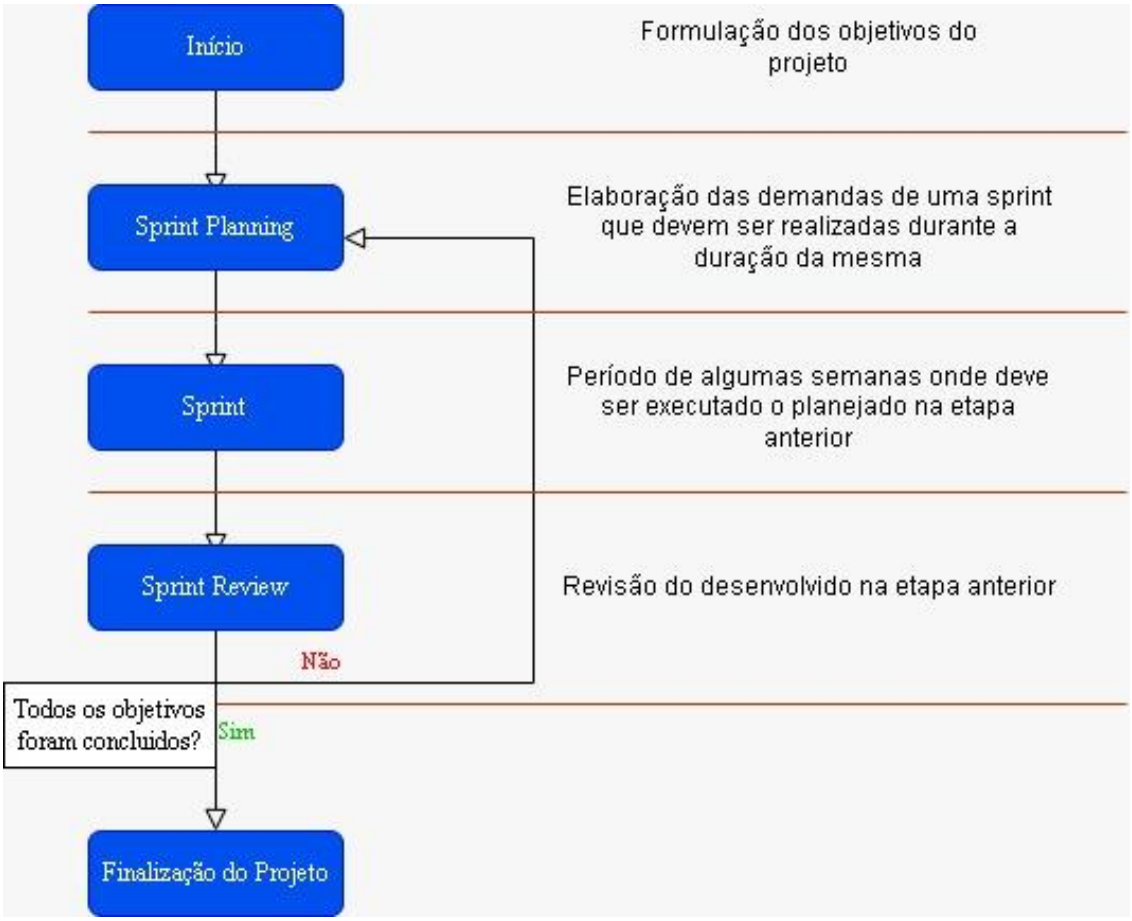
Após a conclusão da *Sprint*, ocorre o *Sprint Review*. Nessa reunião, a equipe de desenvolvimento demonstra o trabalho realizado durante a *Sprint*. Após, é discutido junto dos responsáveis possíveis ajustes e planos para as próximas etapas. O *Sprint Review* é uma peça fundamental para a transparência e a inspeção do trabalho realizado.

Como será utilizado de forma individual, com apenas um colaborador, essa metodologia deve ser adaptada para a obtenção do objetivo. Um exemplo é utilizar o tempo das DS para focar unicamente nos seus objetivos e não discutir com outro. Porém, ainda pode-se ter um *Product Owner,* que é aquele que vai avaliar se o produto atende as suas necessidades especificas.

Pode ser observado na Figura 2 um fluxograma descritivo de um processo de

desenvolvimento através do método Scrum.

Figura 2 – Fluxograma Scrum



Fonte: Autor

3 MATERIAIS E MÉTODOS

Para questão de organização, foi utilizado um caso hipotético de uma indústria de fabricação de painéis fotovoltaicos no Brasil, a título de referência e orientação dos trabalhos.

3.1 Ferramentas Utilizadas

Notebook Acer® Aspire 3

Mouse Genius®

Aparelho Celular *Smart* Xiaomi® Redmi Note 8 Pro

Cabo de transferência de dados USB-C Para USB

3.2 *Softwares* Utilizados

Android Studio® SDK 17.0.6 *Open-Source* (Código Aberto)

Emulador do *Smartphone* Pixel 3 (Ferramenta do Android Studio)

Flutter SDK 13 (Tiramisu) *Open-Source*

Flutter *Tools* 13 (Tiramisu) *Open-Source*

GoogleCloud® API

GoogleSheets®

4 DESENVOLVIMENTO

Nesta seção, está apresentado o desenvolvimento detalhado do aplicativo móvel criado como parte deste trabalho. O aplicativo foi concebido para atender às necessidades específicas identificadas durante a pesquisa e visa fornecer uma alternativa prática e eficaz para a gestão de manutenção. A seguir, estão discutidas as etapas chave do desenvolvimento, incluindo a estrutura das telas e a lógica de programação.

4.1 Estrutura de telas

O aplicativo móvel foi projetado para ser intuitivo e de fácil utilização. Para alcançar esse objetivo, foi dividida a interface do usuário em várias telas distintas, cada uma com funcionalidades específicas. As principais telas incluem:

4.1.1 Tela de cadastro e *login*

Nesta tela, os usuários podem registrar-se no aplicativo, fornecendo informações como e-mail e senha, que são utilizadas para exibir dados específicos de cada usuário. Após o registro, eles podem acessar o aplicativo usando seu e-mail e senha na mesma tela. Pode ser observado no **APÊNDICE A**. Parte do código responsável por gerar a tela do apêndice anterior pode ser observado no **APÊNDICE B**.

4.1.2 Tela setores

Esta tela é exibida após o *login*, permitindo que os usuários adicionem novos setores por meio de um botão *FloatingButton*. Eles também podem selecionar ou excluir setores listados na tela, que são usados para acessar a tela de máquinas ou para excluir um setor desejado. Pode ser observado na **APÊNDICE C**. Parte do código responsável por gerar a tela do apêndice anterior pode ser observado no **APÊNDICE D**.

4.1.3 Tela de máquinas

Nesta tela, os usuários podem adicionar novas máquinas ou acessar as já existentes. As máquinas são exibidas em uma estrutura semelhante a um aplicativo, com ícones e nomes. Além

disso, a tela oferece um *DrawerList* no canto, que fornece opções de navegação para as áreas principais do aplicativo. Pode ser observado em **APÊNDICE E** e **APÊNDICE G**. Parte do código responsável por gerar a tela do apêndice anterior pode ser observado no **APÊNDICE F**.

As funções do *DrawerList* incluem a opção de retornar à tela de setores, adicionar usuários, sair (voltando para a tela de cadastro e *login*) e exibir indicadores.

Tela de detalhes de máquinas: Quando um usuário seleciona uma máquina específica, esta tela exibe detalhes e opções para criar fichas de preenchimento, visualizar fichas preenchidas e, se necessário, excluir fichas e a própria máquina. Essas ações são realizadas por meio de botões *FloatingButtons* abaixo das informações da máquina. Pode ser observado no **APÊNDICE I**. Parte do código responsável por gerar a tela do apêndice anterior pode ser observado no **APÊNDICE J**.

Essa tela permite que, ao apertar o botão, o usuário crie fichas de preenchimento para máquinas desejadas, podendo conter campos de inserção de texto, data e hora. Isto pode ser observado no **APÊNDICE K**.

Ainda na tela de detalhes de máquinas, mostrada no apêndice anterior, o usuário pode preencher as fichas já criadas anteriormente, o que pode ser observado no **APÊNDICE M**. Isto é possível após o usuário selecionar a ficha que queira preencher, como pode ser observado no **APÊNDICE L**.

Isto é possível com a propriedade de *Dialog*, que permite adicionar novos conteúdos nessa espécie de caixa de texto, oportunizando que, assim, que uma tela disponha de múltiplas funções, como se fossem diversas telas diferentes.

Essa tela permite também a visualização organizada dos dados apontados, por intermédio de funções *showDialog*. O botão visualizar ficha de apontamento exibe as fichas criadas e disponibiliza a exclusão, caso desejado. Pode ser observado no **APÊNDICE N** que, quando selecionada uma das fichas, serão exibidos em ordem de criação mais recente para mais antiga esses apontamentos, também disponibilizando a exclusão do apontamento, caso desejado, como é mostrado no **APÊNDICE O**.

A tela em **APÊNDICE N** disponibiliza uma função de exclusão de itens e fichas de apontamento, que podem ser executadas com a ativação do ícone exibido em vermelho. Essa função utiliza da propriedade *ShowConfirmationDialog* do **APÊNDICE T.**

4.1.4 Tela de adicionar usuários

Nesta tela, os usuários podem adicionar ou remover outros usuários que terão acesso ao setor desejado. A seleção do setor desejado para adicionar um novo usuário segue o mesmo método da tela de setores. Pode ser observado no **APÊNDICE P**. Parte do código responsável por gerar a tela do apêndice anterior pode ser observado no **APÊNDICE Q**.

4.1.5 Tela de indicadores

Esta tela permite a visualização dos indicadores do setor e a seleção do MTTR de todas as máquinas ou de uma máquina específica do setor. Pode ser observado no **APÊNDICE R**. Parte do código responsável por gerar a tela do apêndice anterior pode ser observado no **APÊNDICE S**.

Em todos os casos em que existe a possibilidade de exclusão, ao acionar o botão para realizar essa ação, é exibido um *ShowConfirmationDialog*, que solicita a confirmação do usuário antes de prosseguir. Pode ser observado no **APÊNDICE T**.

4.2 Banco de dados

Para armazenar e recuperar todas as informações geradas pelos usuários, foi essencial empregar um banco de dados capaz de gerenciar a transferência de dados pela internet. Nesse sentido, optou-se por utilizar o GoogleCloud®.

O GoogleCloud® API é uma plataforma que possibilita a troca de informações com o GoogleSheets®, um serviço que oferece planilhas online. Atualmente, as informações dos usuários estão sendo armazenadas nesse serviço. Essa escolha baseou-se na facilidade de uso e na clareza visual das informações. Tal abordagem permite a verificação em tempo real da precisão das informações fornecidas pelos usuários e a edição direta na planilha. Isto torna o GoogleSheets® uma ferramenta benéfica para testes durante o desenvolvimento do aplicativo.

4.3 Testes e Ajustes

Durante o processo de desenvolvimento, foram conduzidos testes para que as

funcionalidades do aplicativo fossem ajustadas, conforme a necessidade. Também foi necessário alterar a lógica de algumas telas, com o intuito de atender a necessidade de uma gestão eficiente.

5 TRABALHOS FUTUROS

Após a conclusão do desenvolvimento do aplicativo, tornou-se evidente que existem diversas oportunidades de aprimoramento, englobando tanto o *design* do aplicativo quanto a maneira como os dados são apresentados aos usuários. Neste capítulo, serão exploradas as potenciais melhorias que podem ser implementadas no aplicativo, com o intuito de o tornar mais atraente e eficiente.

5.1 Aperfeiçoamento da visualização de Informações

Nesta seção, são analisadas algumas melhorias potenciais na exibição de informações. As seguintes sugestões têm o potencial de aprimorar a experiência do usuário e tornar a gestão de dados mais eficiente:

5.1.1 Aprimoramento da tela de Indicadores

Uma melhoria pode ser feita na tela de indicadores, permitindo que os usuários filtrem os dados com base nas datas dos apontamentos. Atualmente, essa análise requer uma operação manual e, ao introduzir essa funcionalidade, os usuários podem realizar uma gestão mais eficiente dos dados.

5.1.2 Seleção de fichas

Outra melhoria importante é permitir que os usuários escolham quais fichas desejam visualizar. Isto possibilitará uma análise mais específica quando necessário, atendendo às necessidades individuais dos usuários.

5.1.3 Visualização de apontamentos

Introduzir uma nova tela que permita a visualização de todos os apontamentos de um

setor. Os usuários poderiam filtrar esses apontamentos por data e equipamento, somando todas as visualizações de apontamentos em uma única tela. Essa abordagem simplificaria a gestão e melhoraria a eficiência.

5.2 Melhorias de *design*

Além das melhorias na exibição de informações discutidas anteriormente, aprimorar o aspecto estético do aplicativo é fundamental para mantê-lo atualizado e moderno. A seguir, serão abordadas algumas sugestões de melhorias de *design*:

5.2.1 Atualização de ícones

Existe a possibilidade de atualizar os ícones do aplicativo para um conjunto de ícones mais contemporâneo e alinhado com as últimas tendências de *design*. Ícones modernos podem não apenas melhorar a estética, mas também tornar a navegação mais intuitiva para os usuários.

5.2.2 Revisão do *layout*

Uma revisão do *layout* do aplicativo pode ser realizada. Isto envolve a avaliação e possível reformulação da disposição de elementos e cores, para garantir que o aplicativo tenha uma aparência atraente.

5.2.3 Aprimoramento de *dialogs*

Os *Dialogs* incluem os *showConfirmationDialog e showDialog,* que desempenham um papel crucial na interação com os usuários. Aprimorar o *design* dessas caixas de diálogo poderia torná-las mais atraentes e amigáveis. Isto pode incluir o uso de animações sutis e uma estética que se alinhe com a identidade visual do aplicativo.

5.2.4 Paleta de cores

Reavaliar a paleta de cores do aplicativo pode ser uma melhoria significativa. A seleção de cores pode impactar diretamente na percepção do usuário e na usabilidade.

Estas são algumas das melhorias consideradas para aprimorar a exibição de informações

no aplicativo, proporcionando maior flexibilidade e controle aos usuários.

6 CONSIDERAÇÕES FINAIS

Durante este trabalho, foi possível explorar e analisar as capacidades de desenvolvimento que a linguagem Dart oferece. Essa exploração resultou no desenvolvimento de um aplicativo que não apenas armazena informações importantes para a gestão de manutenção, mas também simplifica sua análise, fornecendo um auxílio valioso aos gestores nessa área.

No entanto, o desenvolvimento não é um processo estático. Há sempre espaço para melhorias contínuas. Com o contínuo aprimoramento, o aplicativo tornar-se-á uma ferramenta útil, na área de gestão de manutenção.

Este trabalho destaca a importância da inovação e da tecnologia, no contexto da gestão de manutenção. A linguagem Dart e o aplicativo resultante demonstram isso.

A capacidade de armazenar informações em tempo real em um banco de dados oferece às equipes de manutenção a possibilidade de monitorar o desempenho dos equipamentos e detectar potenciais problemas. Isto pode resultar em menos tempo de inatividade não planejado e na redução de custos operacionais.

REFERÊNCIAS BIBLIOGRÁFICA

AQEEL, AbdulMuaz. **Flutter Clean Architecture Series**. 2021. Disponível em: https://devmuaz.medium.com/flutter-clean-architecture-series-part-1-d2d4c2e75c47. Acesso em: 02 nov. 2023.

BIESSEK, Alessandro. **Flutter for beginners**.Packt Publishing, 2019. Páginas 122-140.

CABRAL, J. (2009). **Gestão da manutenção de equipamentos, instalações e edifícios**. Biblioteca Industria & Serviços, LIDEL, Abril. ISBN: 9789727579709.
CARVALHO, Maria Ana dos Santos**.** Proposta de Modelo para a Melhoria Contínua das Atividades de Gestão da Manutenção. 2019. Disponível em: https://run.unl.pt/bitstream/10362/96115/1/Carvalho_2019.pdf. Acesso em: 01 de junho 2023.

DAGNE, Lukas. **Flutter for cross-platform App and SDK development.** Trabalho de conclusão de curso (Bachelor of Engineering) - Metropolia University of Applied Sciences, 2019. Disponível em: https://www.theseus.fi/bitstream/handle/10024/172866/Lukas%20Dagne%20Thesis.pdf. Acesso em: 04 de maio de 2023.

FAUST, Sebastian. **Using Google's Flutter Framework for the Development of a Large-Scale Reference Application**. 2020. Disponível em: https://epb.bibl.th-koeln.de/frontdoor/index/index/docId/1498. Acesso em: 28 jan. 2023.

FRANKLIN, Matheus Maião; FILHO, Ronaldo Aparecido Samuel. **Desenvolvimento de um Sistema de Gestão Escolar com o uso da Linguagem Dart com Framework** Flutter. 2020. Disponível em: http://ric.cps.sp.gov.br/handle/123456789/7070. Acesso em: 12 jun. 2023.

FREITAS, E. (2019). **Xamarin.Forms Succinctly. Syncfusion, Inc.** Disponível em: http://www.syncfusion.com/. Acesso em: 12 jun. 2023.

FREITAS, Sérgio Augusto. **Aumento na eficiência de equipamentos com o uso da ferramenta de qualidade análise de falhas.** 2018. Disponível em:

https://www.unifacvest.edu.br/assets/uploads/files/arquivos/67446-tcc-sergio-augusto-freitas-eng.-mecanica-2018.pdf.. Acesso em: 29 jun. 2023.

FUENTES, F. F. E. **Metodologia para inovação da gestão de manutenção industrial.** 2006. Disponível em: https://repositorio.ufsc.br/handle/123456789/88894. Acesso em: 4 maio 2023.

Fundação Getulio Vargas (FGV). **Brasil tem 424 milhões de dispositivos digitais em uso, revela a 31ª Pesquisa Anual do FGVcia**. 2020. Disponível em: < https://portal.fgv.br/noticias/brasil-tem-424-milhoes-dispositivos-digitais-uso-revela-31a-pesquisa-anual-fgvcia >. Acesso em: 02 nov. 2023.

MATOS, Beatriz Rezener Dourado; SILVA, João Gabriel de Britto e. **Estudo comparativo entre o desenvolvimento de aplicativos móveis utilizando plataformas nativas e multiplataforma. 2016.** Disponível em: https://fga.unb.br/articles/0001/5114/Beatriz_Joao_TCC_Aplicativos_M_veis.pdf. Acesso em: 4 maio 2023.

MORGAN, Brett; LOUGHEED, Parker. **FAQ Flutter**. [S.l.], 2023. Disponível em: https://docs.flutter.dev/resources/faq. Acesso em: 28 out. 2023.

OLIVEIRA, George Souza; SILVA, Anderson Faustino da. **Compilação Just-In-Time: Histórico, Arquitetura, Princípios e Sistemas**. In: Universidade Estadual de Maringá. Revista de Informática Teórica e Aplicada, v. 20, n. 2, 2013. Disponível em: https://seer.ufrgs.br/rita/article/view/rita_v20_n2_p155WesleyVol20Nr2_174/25446. Acesso em: 31 de maio de 2023.

OTANI, Mario; MACHADO, Waltair Vieira. **Aumento na eficiência de equipamentos com o uso da ferramenta de qualidade análise de falhas.** 2018. Disponível em: https://www.unifacvest.edu.br/assets/uploads/files/arquivos/67446-tcc-sergio-augusto-freitas-eng.-mecanica-2018.pdf. Acesso em: 29 jun. 2023.

PINTO, A. K; XAVIER, J. A. N. **Manutenção: função estratégica.** Rio de Janeiro: Qualitymark, 2001. Disponivel em: http://ftp.demec.ufpr.br/disciplinas/TM285/2015-2/Conte%FAdos/Resumo%20Livro%20Manuten%E7%E3o.pdf. Acesso em: 04 mai. 2023.

POMBO, Rocha. **Dicionário de Sinônimos da Língua Portuguesa**. 2. ed. Rio de Janeiro: Academia Brasileira de Letras, 2011. Disponível em: https://www.academia.org.br/sites/default/files/publicacoes/arquivos/cams-10-dicionario_de_sinonimos_da_lingua_portuguesa-para_internet.pdf. Acesso em: 02 nov. 2023.

PREZOTTO, E. D., & Boniati, B. B. 2014. **Estudo de frameworks multiplataforma para desenvolvimento de aplicações mobile híbridas.** Disponível em: https://docplayer.com.br/16198154-Estudo-de-frameworks-multiplataforma-para-desenvolvimento-de-aplicacoes-mobile-hibridas.html. Acesso em: 22 abr. 2023.

RAMOS, Pedro Gonçalo Diniz. **Organização e Gestão da Manutenção Industrial: Aplicação Teórico prática às Fabricas Lusitana Produtos Alimentares, S.A.** 2012. Disponível em: https://www.proquest.com/openview/48fd1caad02598b1b43f9696adf7c75d/1?pq-origsite=gscholar&cbl=2026366&diss=y. Acesso em: 04 de maio de 2023.

ROSA, E. B. 2006. **Indicadores de desempenho e sistema ABC: o uso de indicadores para uma gestão eficaz do custeio e das atividades de manutenção.** Disponível em: https://www.feg.unesp.br/Home/PaginasPessoais/profsalomon/eurycibiadesbarrarosa.pdf. Acesso em: 01 de junho 2023.

ROSCOFF, Nathália Santos; COSTELLA, Marcelo Fabiano; PILZ, Silvio Edmundo. **Desenvolvimento de *software* para gestão da manutenção preventiva em edificações habitacionais.** ENTAC - Encontro Nacional de Tecnologia do Ambiente Construído, 2020. Disponível em: https://eventos.antac.org.br/index.php/entac/article/view/1160. Acesso em: 04 de maio de 2023.

SILVA, Marcos Antonio Alves da. **Desenvolvimento de um aplicativo para gerenciamento da manutenção do sistema elétrico de potência da UFRN, utilizando o código QR como ferramenta de acesso ao plano de manutenção**. 2017. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Rio Grande do Norte, Natal, 2017. Disponível em: https://repositorio.ufrn.br/handle/123456789/24486. Acesso em: 09 abr. 2023.

Starcounter. **Mobile Operating System Market Share Worldwide**. 2023. Disponível em: https://gs.statcounter.com/os-market-share/mobile/worldwide. Acesso em: 28 out. 2023.

STOLL, Scott. **In plain English: So what the heck is Flutter and why is it a big deal?**. 2018. Disponível em: https://medium.com/flutter-community/in-plain-english-so-what-the-heck-is-flutter-and-why-is-it-a-big-deal-7a6dc926b34a. Acesso em: 28 out. 2023.

TeleGeography®. Disponível em: < https://www2.telegeography.com>. Acesso em: 09 abr. 2023.

WINDMILL, Eric. "**Flutter In Action**". Prefácio por Ray Rischpater. Manning Publications Co., 2020.

XENOS, Harilaus G. **Gerenciando a Manutenção Produtiva**, Belo Horizonte: editora DG, 1998. Disponivel em: https://pt.scribd.com/doc/150319707/Livro-Gerenciando-a-Manutencao-Produtiva, Acesso em: 04 mai. 2023.

YAMAKAWA, Eduardo Kazumi; MIGUEL, Paulo Augusto Cauchick; AOKI, Alexandre Rasi. **Aplicação de Fuzzy Quality Function Deployment para seleção de indicadores de eficiência energética para utilização em um *software* de gestão de energia.** Science & Engineering Journal, 2014. Disponível em: <https://www.researchgate.net/profile/Eduardo-Yamakawa/publication/276415975_Aplicacao_de_Fuzzy_Quality_Function_Deployment_para_selecao_de_indicadores_de_eficiencia_energetica_para_utilizacao_em_um_software_de_gestao_de_energia/links/56cc5e4608ae1106370d89a1/Aplicacao-de-Fuzzy-Quality-Function-Deployment-para-selecao-de-indicadores-de-eficiencia-energetica-para-utilizacao-em-um-software-de-gestao-de-energia.pdf>. Acesso em: 04 mai. 2023.

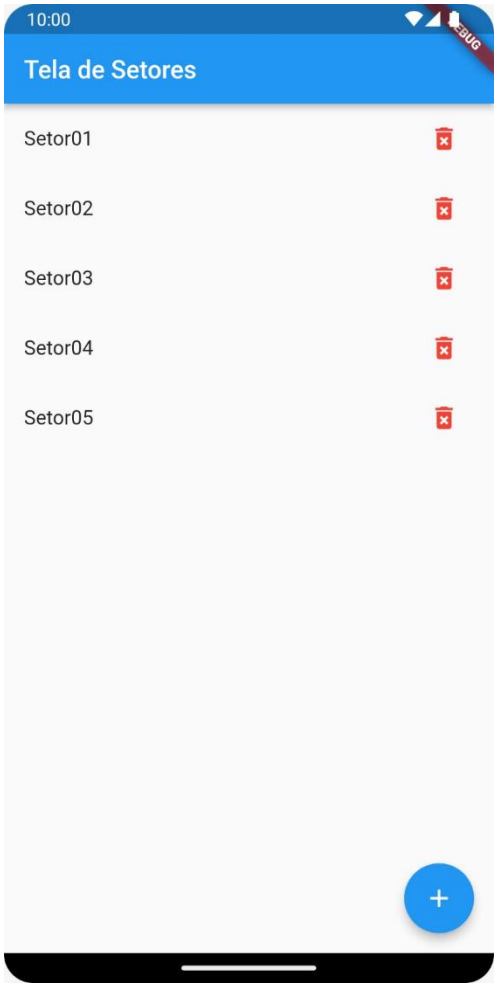
YOU, Dongliang; HU, Minjie. **A Comparative Study of Cross-platform Mobile Application Development**. 2021. Disponível em: https://www.researchgate.net/publication/357898491_A_Comparative_Study_of_Cross-platform_Mobile_Application_Development. Acesso em: 28 out. 2023.

APÊNDICE A — Figura da tela de cadastro e *login*

APÊNDICE B — Código de tela de cadastro e *login*

```
Expanded(
  flex: 7,
  child: Padding(
    padding: EdgeInsets.all(16.0),
    child: Column(
      children: [
        TextField(
          controller: _loginController,
          decoration: InputDecoration(
            labelText: 'Email',
          ),  // InputDecoration
        ),  // TextField
        SizedBox(height: 10.0),
        TextField(
          controller: _passwordController,
          obscureText: true,
          decoration: InputDecoration(
            labelText: 'Senha',
          ),  // InputDecoration
        ),  // TextField
        if (_showConfirmPassword)
          SizedBox(height: 10.0),
        TextField(
          controller: _confirmPasswordController,
          obscureText: true,
          decoration: InputDecoration(
            labelText: 'Confirmar Senha',
```

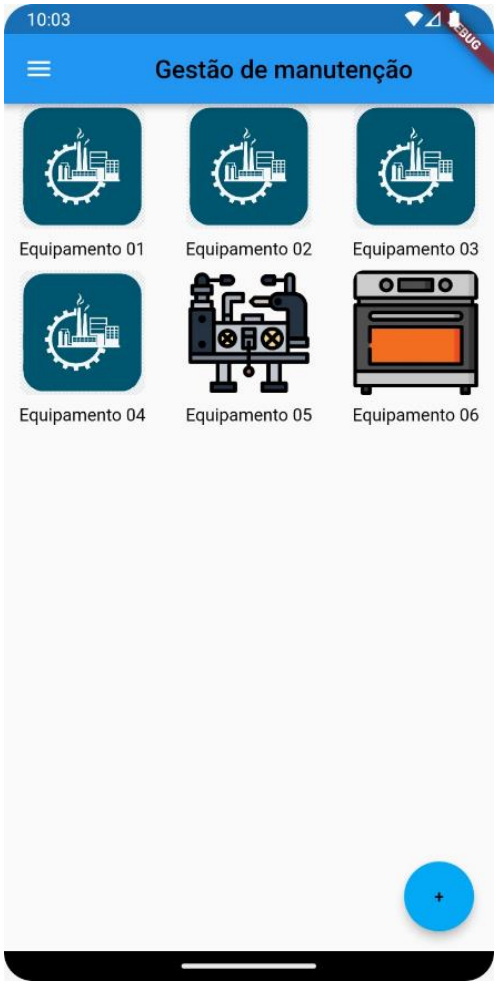
APÊNDICE C — Figura da de tela setores



APÊNDICE D — Código de tela setores

```
Widget build(BuildContext context) {
  return AlertDialog(
    title: Text("Adicionar Setor"),
    content: TextField(
      controller: _nomeController,
      decoration: InputDecoration(labelText: "Nome do Setor"),
    ),  // TextField
    actions: [
      TextButton(
        onPressed: () {
          Navigator.pop(context);
        },
        child: Text("Cancelar"),
      ),  // TextButton
      TextButton(
        onPressed: () {
          String nome = _nomeController.text.trim();
          if (nome.isNotEmpty) {
            Navigator.pop(context, nome);
          }
        },
        child: Text("Adicionar"),
      ),  // TextButton
    ],
  );  // AlertDialog
}
```

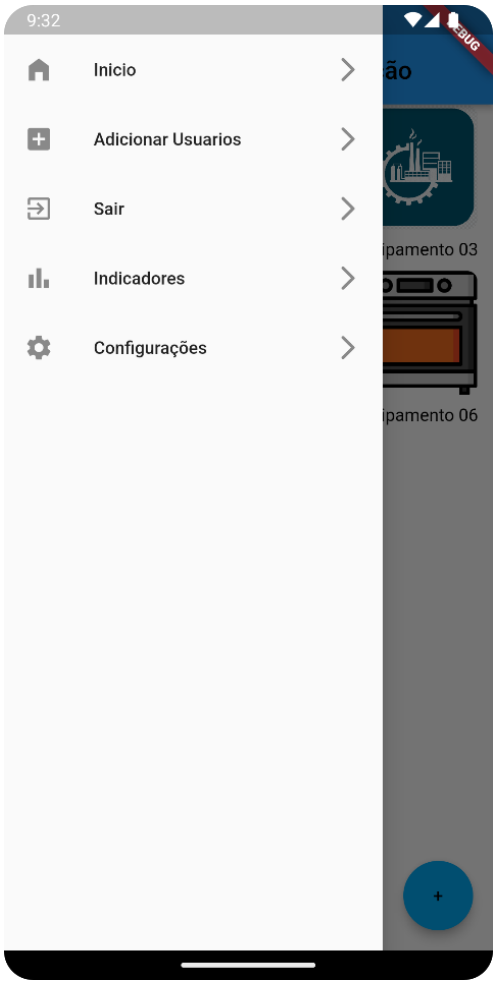
APÊNDICE E — Figura da de tela máquinas



APÊNDICE F — Código de tela máquinas

```
child: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Container(
      constraints: BoxConstraints(maxHeight: 100, maxWidth: 100),
      child: ClipRRect(
        borderRadius: BorderRadius.circular(8.0),
        child: Image.network(
          selectedMachines[index].url,
          fit: BoxFit.cover,
        ),  // Image.network
      ),  // ClipRRect
    ),  // Container
    SizedBox(height: 8),
    Text(
      selectedMachines[index].machine,
      textAlign: TextAlign.center,
      style: TextStyle(color: Colors.black),
```

APÊNDICE G — Figura da DrawerList



APÊNDICE H — Código de DrawerList

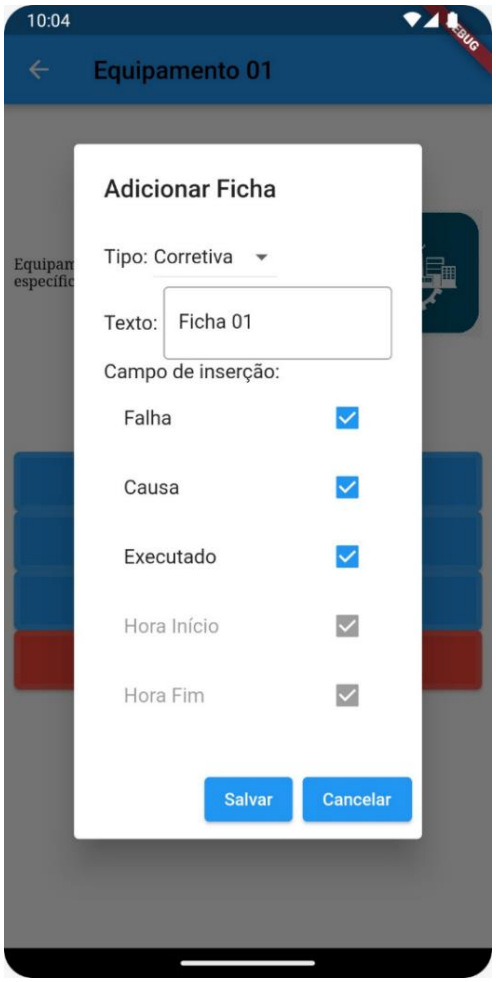
```
return Drawer(
  child: ListView(
    children: <Widget>[
      ListTile(
        leading: Icon(Icons.home_filled),
        title: Text("Inicio"),
        trailing: Icon(Icons.arrow_forward_ios_rounded),
        onTap: () {
          _onClickedSetores(context);
          print("Inicio");
        },
      ),  // ListTile
      ListTile(
        leading: Icon(Icons.add_box),
        title: Text("Adicionar Usuarios"),
        trailing: Icon(Icons.arrow_forward_ios_rounded),
        onTap: () {
          _onClickedaddUsers(context);
          print("Adicionar Usuarios");
        },
      ),  // ListTile
      ListTile(
        leading: Icon(Icons.exit_to_app_outlined),
        title: Text("Sair"),
        trailing: Icon(Icons.arrow_forward_ios_rounded),
        onTap: () {
          _onClickedSair(context);
          print("Sair");
        },
      ),  // ListTile
```

APÊNDICE I — Figura da tela detalhes de máquinas

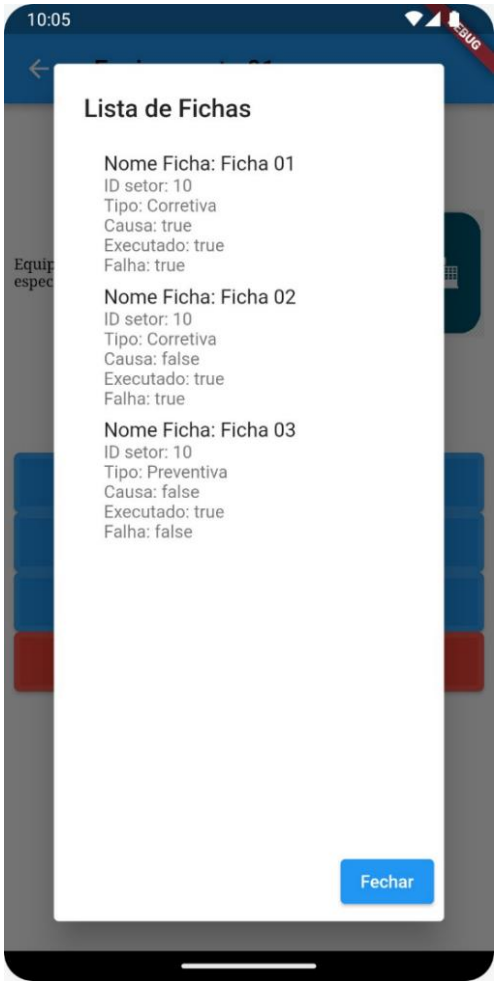
APÊNDICE J — Código de tela detalhes de máquinas

```
return Scaffold(
  appBar: AppBar(
    title: Text(machine.machine, style: TextStyle(color: Colors.black)),
  ),  // AppBar
  body: Column(
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: [
      Expanded(
        flex: 2, // Ajuste para 20% da tela
        child: Padding(
          padding: EdgeInsets.all(8.0),
          child: Row(
            children: [
              Expanded(
                flex: 8, // Ajuste para 80% da tela
                child: Text(
                  machine.descricao,
                  style: TextStyle(color: Colors.black, fontFamily: 'Times New Roman', fontSize: 12),
                ),  // Text
              ),  // Expanded
              Expanded(
                flex: 3,
                child: Image.network(machine.url),
```

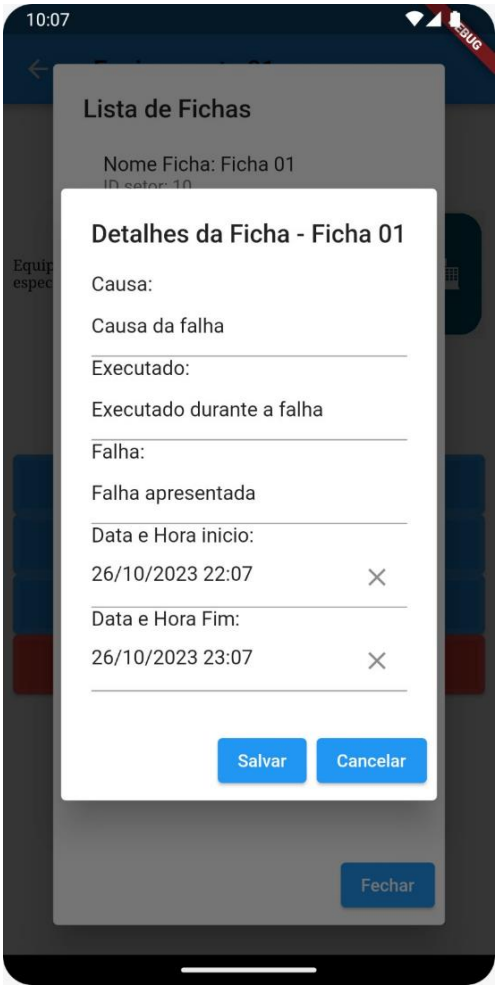
APÊNDICE K — Figura da tela detalhes de máquinas



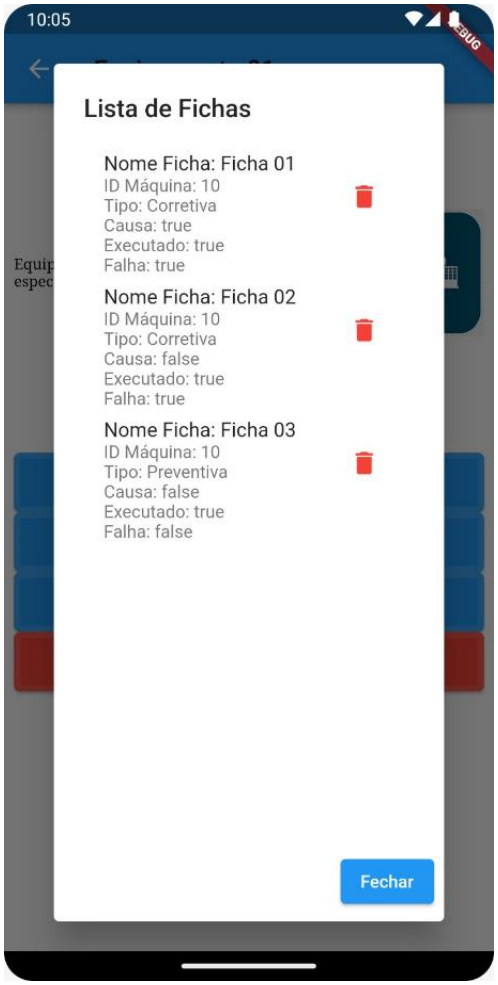
APÊNDICE L — Tela de seleção de ficha



APÊNDICE M — Tela para adicionar apontamento

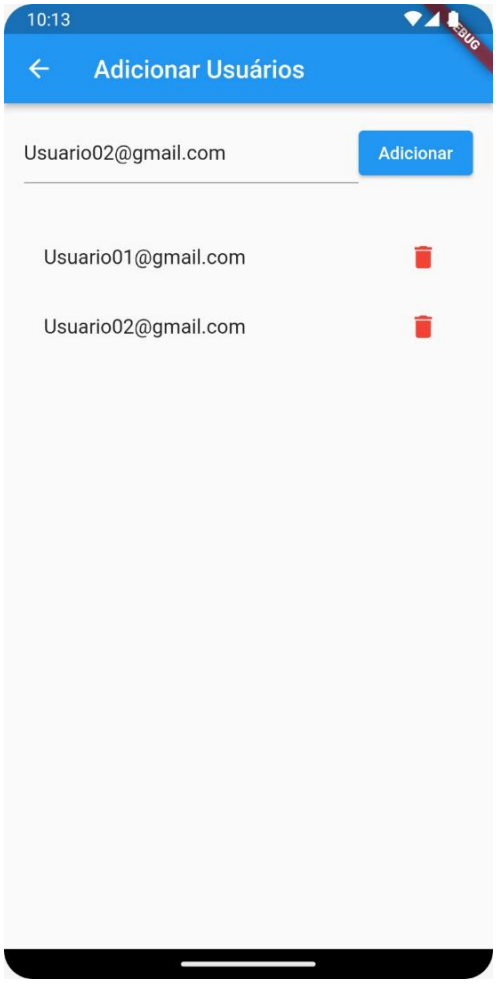


APÊNDICE N — Tela para visualizar fichas de apontamento



APÊNDICE O — Tela para visualizar apontamento

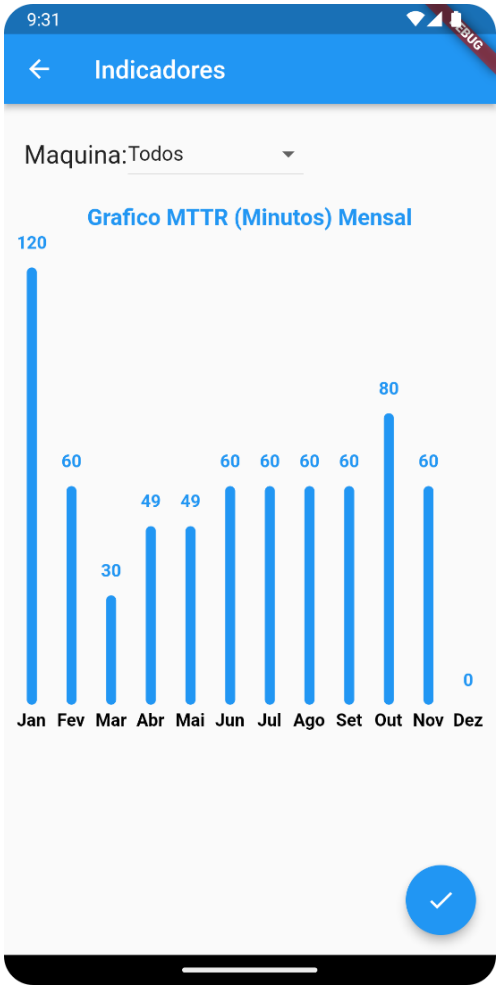
APÊNDICE P — Tela para adicionar usuários



APÊNDICE Q — Código de tela adicionar usuários

```
body: SingleChildScrollView(
  child: Column(
    children: <Widget>[
      Padding(
        padding: const EdgeInsets.all(16.0),
        child: Row(
          children: <Widget>[
            Expanded(
              child: TextField(
                onChanged: (value) {
                  setState(() {
                    newUser = value;
                  });
                },
                decoration: InputDecoration(
                  hintText: 'Digite o nome do usuário',
                ),  // InputDecoration
              ),  // TextField
            ),  // Expanded
            ElevatedButton(
              onPressed: () {
                if (newUser.isNotEmpty) {
                  setState(() {
                    _addUser(newUser, setorId);
                    newUser = '';
                  });
                }
              },
```

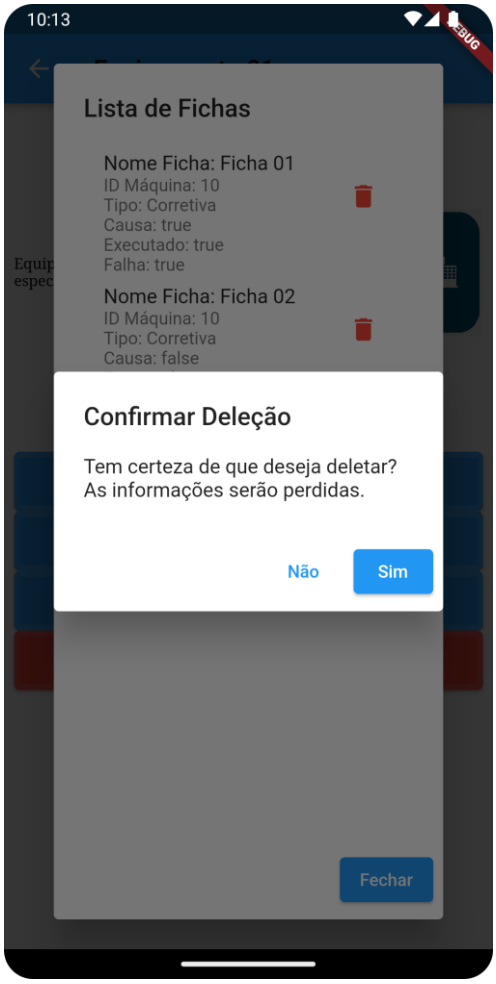
APÊNDICE R — Tele de indicadores



APÊNDICE S — Código da tela Indicadores

```
Text(
  "Grafico MTTR (Minutos) Mensal",
  style: TextStyle(
    fontSize: 18,
    fontWeight: FontWeight.bold,
    color: Colors.blue,
  ),  // TextStyle
),  // Text
SizedBox(
  height: 30, // Espaço de 16 pixels entre o texto e o gráfico
),  // SizedBox
Container(
  height: 380,
  width: 380,
  child: _BarChart(
    totalHorasJaneiro: totalHorasJaneiro * 1440,
    totalHorasFevereiro: totalHorasFevereiro * 1440,
    totalHorasMarco: totalHorasMarco * 1440,
    totalHorasAbril: totalHorasAbril * 1440,
    totalHorasMaio: totalHorasMaio * 1440,
    totalHorasJunho: totalHorasJunho * 1440,
    totalHorasJulho: totalHorasJulho * 1440,
```

APÊNDICE T — Figura de *ShowConfirmationDialog*



CENTRO UNIVERSITÁRIO

DECLARAÇÃO DE REVISÃO
ORTOGRÁFICA E GRAMATICAL

Eu, **JORGE ANTONIO BERNDT**, RG **14627664-4**, CPF **090.159.019-37**, e-mail **JORGEBERNDT@ICLOUD.COM**, telefone **(45) 3039-2335**, declaro para os devidos fins que fiz a correção ortográfica e gramatical do Trabalho de Conclusão de Curso intitulado DESENVOLVIMENTO DE SOFTWARE VOLTADO À GESTÃO DE MANUTENÇÃO, de autoria de Daniel Zago Dias, acadêmico(a) regularmente matriculado no Curso de Engenharia Elétrica do Centro Universitário FAG.

Cascavel, 13 de dezembro de 2023.

Jorge Antonio Berndt

1º TABELIONATO DE NOTAS

JORGE ANTONIO BERNDT
RG: **14627664-4**
CPF: **090.159019-37**

1º TABELIONATO DE NOTAS - CASCAVEL - PARANÁ
RUA SOUZA NAVES 3755 - CASCAVEL - PARANÁ - CEP 85801-120 - FONE (45) 2101-7963
FERNANDO CESAR VELLOZO LUCASKI - TABELIÃO

Selo Nº SFTN1vGHYbO8sunmKhmdF229q - Valide em funarpen.com.br. Reconheço por SEMELHANÇA, a assinatura de JORGE ANTONIO BERNDT (371739). Cascavel-PR, 13 de dezembro de 2023 01:30:15:30:19*

Em Test° da Verdade
Graziele Massaneiro de Souza - Escrevente

1º TABELIONATO DE NOTAS
Fernando Cesar Vellozo Lucaski
Tabelião
Fone: (45) 2101-7963
Comarca de Cascavel-PR

CASCAVEL-PR

CENTRO UNIVERSITARIO DA FUNDAÇÃO ASSIS GURGACZ

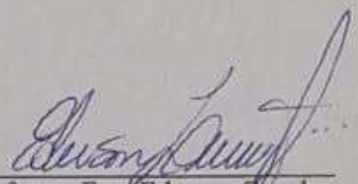
DANIEL ZAGO DIAS

DESENVOLVIMENTO DE SOFTWARE VOLTADO A GESTÃO DE MANUTENÇÃO

Trabalho apresentado no Curso de Engenharia Elétrica, do Centro Universitário Assis Gurgacz, como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica, sob a orientação do Professor Mestre Elenilton Dezengrini.

BANCA EXAMINADORA

Professor Me. Elenilton Dezengrini
Centro Universitário da FAG



Professor Esp. Ederson Zanchet
Centro Universitário da FAG

Professor Me. Helder José Costa Carozzi
Centro Universitário da FAG

Cascavel/PR., 21 de novembro de 2023.