Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

Desenvolvimento de um protótipo multiplataforma (*Windows* e *Android*) para apoio às atividades agrárias

Rodrigo Ribeiro de Souza

Prof. Dr. Evandro Cesar Bracht (Orientador)

Dourados – MS
2025

Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

Desenvolvimento de um protótipo multiplataforma (*Windows* e *Android*) para apoio às atividades agrárias

Rodrigo Ribeiro de Souza

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Rodrigo Ribeiro de Souza e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Dourados, 21 de outubro de 2025

Prof. Dr. Evandro Cesar Bracht
(Orientador)

Dourados – MS
2025

S718d Souza, Rodrigo Ribeiro de.

Desenvolvimento de um protótipo multiplataforma (Windows e Android) para apoio às atividades agrárias / Rodrigo Ribeiro de Souza. – Dourados, MS: UEMS, 2025.

63 p.

Monografia (Graduação) – Sistemas de Informação – Universidade Estadual de Mato Grosso do Sul, 2025.

Orientador: Prof. Dr. Evandro Cesar Bracht.

1. Protótipo 2. Tecnologia agrária 3. Agricultura de precisão 4. Software de aplicação I. Bracht, Evandro Cesar II. Título

CDD 23. ed. - 005.28

Ficha Catalográfica elaborada pela bibliotecária da Universidade Estadual de Mato Grosso do Sul (UEMS)
Bruna Peruffo Vieira – CRB 1/2959

Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

Desenvolvimento de um protótipo multiplataforma (*Windows* e *Android*) para apoio às atividades agrárias

Rodrigo Ribeiro de Souza

Outubro de 2025

Banca Examinadora:

Prof. Dr. Evandro Cesar Bracht (Orientador)
Área de Computação – UEMS

Prof. Dr. Diogo Fernando Trevisan
Área de Computação – UEMS

Prof. Dr. Ricardo Luís Lachi
Área de Computação – UEMS

Dourados – MS
2025

AGRADECIMENTOS

Dedico este trabalho primeiramente a Deus, por me conceder forças, sabedoria e saúde para chegar até aqui.

À minha mãe, Maria Valdelice Hora, que mesmo diante de tantas dificuldades, nunca mediu esforços para me educar e cuidar de mim, sendo exemplo de força, dedicação e amor incondicional.

Aos meus amigos, pela amizade sincera, pelas risadas compartilhadas e por estarem presentes em cada etapa dessa caminhada.

E à minha esposa, Thais Castro, pelo amor, dedicação e companheirismo, que me sustentaram nos momentos difíceis e tornaram esta caminhada mais leve, feliz e possível.

A todos, deixo registrado meu sincero reconhecimento e profunda gratidão.

Rodrigo Ribeiro de Souza

RESUMO

Este trabalho apresenta um protótipo de um sistema multiplataforma voltado para auxiliar nas atividades agrárias, especialmente direcionado a pequenos e médios produtores que não dispõem de condições financeiras para investimentos em tecnologias de alto custo, mas que necessitam de melhorias em seus planejamentos e operações. O protótipo foi implementado em *Dart/Flutter*, com distribuição para *Android* e *Windows*, e utiliza uma arquitetura em nuvem baseada no *Supabase* (*PostgreSQL*) para persistência e sincronização dos dados. Foram incorporadas tecnologias de geolocalização, GPS, que possibilitam coletar e registrar as coordenadas dos trajetos percorridos por máquinas agrícolas durante a execução das operações no campo. Esses dados são armazenados e apresentados em relatórios diários, fornecendo uma visão detalhada das atividades realizadas e permitindo, ainda, o rastreamento em tempo real dos equipamentos, o que contribui para a otimização das operações agrícolas. A metodologia adotada envolveu revisão bibliográfica, análise das tecnologias disponíveis no mercado e desenvolvimento de um protótipo funcional. Os resultados obtidos indicam que a solução proposta pode reduzir custos, aumentar a eficiência nas atividades agrícolas e democratizar o acesso à tecnologia, promovendo maior competitividade e reduzindo a desigualdade tecnológica entre produtores de diferentes portes.

Palavras-chave: *tecnologia agrária, agricultura de precisão, software de aplicação, protótipo, monitoramento agrário.*

ABSTRACT

This work presents a prototype of a multiplatform system designed to support agricultural activities, especially for small and medium-sized producers who lack the financial resources to invest in high-cost technologies but require improvements in planning and operations. The prototype was implemented in Dart/Flutter, distributed for Android and Windows, and uses a cloud architecture based on *Supabase* (PostgreSQL) for data persistence and synchronization. Geolocation and GPS technologies were incorporated to collect and record the coordinates of routes traveled by agricultural machinery during field operations. These data are stored and presented in daily reports, providing a detailed view of the activities performed and enabling real-time tracking of equipment, which contributes to optimizing agricultural operations. The adopted methodology involved a literature review, analysis of technologies available on the market, and the development of a functional prototype. The results indicate that the proposed solution can reduce costs, increase efficiency in agricultural activities, and democratize access to technology, promoting greater competitiveness and reducing the technological gap among producers of different sizes.

***Keywords**: agrarian technology, precision agriculture, application software, prototype, agricultural monitoring.*

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

1. Introdução

As tecnologias, como agricultura de precisão, geoprocessamento, monitoramento e *Internet* das Coisas (IoT), entre outras existentes no mercado, vêm se tornando cada vez mais essenciais no setor agrário, contribuindo para o aumento da produtividade, redução dos custos das operações e a minimização dos impactos ambientais. No entanto, pequenos e médios produtores ainda enfrentam dificuldades para adquirir essas ferramentas devido ao alto custo de implantação, manutenção e complexidade operacional. Ou seja, os *softwares* de gestão agrícola existentes no mercado geralmente apresentam custos elevados e exigem arquiteturas computacionais avançadas, o que eleva ainda mais o custo final. Esse cenário reforça a necessidade de uma ferramenta tecnológica acessível que auxilie no planejamento e controle das atividades agrárias.

Diante desse contexto, este trabalho apresenta um protótipo multiplataforma, voltado para auxiliar nas atividades agrárias, com suporte a *Windows* e *Android*. A solução utiliza recursos acessíveis, como geolocalização e GPS (*Global Positioning System*), para oferecer apoio à gestão e ao rastreamento de máquinas agrícolas. O protótipo oferece suporte nos seguintes aspectos: gestão e rastreamento de máquinas agrícolas, no planejamento das operações e na melhoria da produtividade. O desenvolvimento de aplicações tecnológicas voltadas para agricultores de pequeno e médio porte pode contribuir para a modernização da agricultura, promovendo eficiência, sustentabilidade e competitividade no setor agrário, reduzindo desigualdades tecnológicas entre produtores de diferentes portes.

Nas próximas subseções, são apresentados a justificativa, os objetivos e a metodologia adotada para o desenvolvimento deste protótipo.

1.1 Justificativa

Dentre os diversos produtos disponíveis no mercado, podemos destacar os sistemas oferecidos pela empresa (TOTVS, 2021), como o PIMS (*Plant Information Management System*), também conhecido como Sistema de Gerenciamento de Informação de Planta, e os serviços da RR Tecnologia Agrícola (Realmente Resolve), especializada em agricultura de precisão, projetos de pulverização e plantio, além de consultoria técnica em GPS agrícola e piloto automático (RR TECNOLOGIA AGRÍCOLA, 2023). Ambos possuem sistemas complexos e exigem uma estrutura operacional planejada com configuração específica para funcionar corretamente, envolvendo profissionais da computação e da agricultura, o que eleva as despesas de implantação. Além disso, quando implantados, geram custos adicionais com consultoria, manutenção e eventuais assinaturas, aumentando ainda mais o valor final. Como resultado, o preço desses sistemas agrícolas torna-se elevado, dificultando o acesso de pequenos e médios produtores, que ficam impossibilitados de usufruir dessas tecnologias.

Diante desse cenário, torna-se necessária uma aplicação multiplataforma, executável em *Windows* e *Android*, que utilize um banco de dados de baixo custo compartilhado e não dependa de servidores dedicados nem de instalações complexas. Assim, as configurações e o gerenciamento podem ser realizados em um *desktop* com *Windows*, enquanto a coleta das informações ocorre em dispositivos *Android* (*smartphones* ou *tablets*) em campo, com sincronização para o mesmo banco de dados. A ideia é que a aplicação possa se conectar facilmente a uma estrutura simples de *internet* via satélite, como a oferecida pela *Starlink*, desenvolvida pela *SpaceX*. Dessa forma, a aplicação proposta poderá ser instalada em dispositivos compatíveis com *Android*, como *smartphones* e *tablets*, aproveitando ainda a tecnologia GPS já integrada nesses equipamentos, eliminando a necessidade de assinaturas adicionais e reduzindo significativamente os custos para o usuário.

1.2 Objetivos

Nesta seção são descritos os objetivos que orientam o desenvolvimento do protótipo. O objetivo geral apresenta a finalidade principal do trabalho, voltada para uma solução multiplataforma de apoio às atividades agrícolas, com configuração e

gestão no *Windows* e coleta de dados em campo no *Android*, compartilhando um banco de dados de baixo custo e fácil implantação.

1.2.1 Objetivo geral

O objetivo principal desta pesquisa é desenvolver um aplicativo de baixo custo voltado para o setor agrícola, que permita ao produtor rural coletar dados digitalmente e, assim, obter relatórios mais consistentes, possibilitando a melhoria no planejamento, no rendimento e no controle de custo. Além disso, o aplicativo utilizará a tecnologia GPS para identificar se os equipamentos estão posicionados corretamente nas áreas de manejo.

Para tanto temos que desenvolver os seguintes objetivos específicos:

- Levantar os requisitos necessários para desenvolvimento do protótipo;
- Definir a implementação do banco de dados;
- Gerar relatórios estruturados com os dados coletados pelo aplicativo, a fim de facilitar a visualização e apoiar a tomada de decisão.

1.3 Metodologia

A metodologia empregada neste trabalho inclui tanto a revisão bibliográfica sobre o uso de tecnologias digitais aplicadas ao setor agrícola quanto o desenvolvimento de um protótipo funcional multiplataforma (*Windows* e *Android*).

A primeira etapa consistiu no levantamento de requisitos, que serviu para identificar as necessidades dos usuários e definir as principais funcionalidades do protótipo. Em seguida, foi realizada a análise das ferramentas disponíveis para o desenvolvimento do protótipo, com ênfase no uso da linguagem de programação *Dart* e do *framework Flutter*, por se tratarem de tecnologias que permitem a criação de aplicações multiplataforma compatíveis com o sistema operacionais *Windows* e *Android* (Google, 2025; Dart, 2025).

Na etapa seguinte, foi definida a estrutura do banco de dados, de forma a possibilitar o registro das informações coletadas pelo usuário. O protótipo também integrou a funcionalidade de geolocalização por GPS.

Após a implementação, foram realizados testes funcionais para verificar a

usabilidade, a confiabilidade e a performance do protótipo em diferentes dispositivos *Android*. Por fim, os dados obtidos foram organizados em relatórios estruturados, possibilitando ao usuário melhor visualização das informações e apoio ao processo de tomada de decisão.

Dessa forma, a metodologia apresentada orienta a organização das próximas seções deste trabalho. No **Capítulo 2**, são discutidos os fundamentos teóricos que sustentam o estudo, abrangendo tecnologias digitais aplicadas ao meio agrícola. O **Capítulo 3** descreve o processo de desenvolvimento do protótipo, contemplando as tecnologias utilizadas, os requisitos funcionais, a modelagem do banco de dados e implementação das funcionalidades. Por fim, o **Capítulo 4** apresenta as conclusões do estudo e as perspectivas para trabalhos futuros. Assim, a estrutura do texto reflete de forma direta as etapas metodológicas adotadas.

2. Revisão bibliográfica

O uso de tecnologias no setor agrícola tem sido essencial para melhorar a eficiência das operações e aumentar a produtividade nas atividades agrárias. Com a crescente demanda por soluções mais precisas e sustentáveis, diversas ferramentas tecnológicas têm se destacado, proporcionando aos agricultores maior controle e otimização de seus recursos.

Neste capítulo, são apresentadas algumas das tecnologias aplicadas ao setor agrário, suas principais funções e como contribuem para um gerenciamento mais eficiente e sustentável das atividades agrícolas. Inicialmente, aborda-se o conceito de agricultura de precisão e sua relação com o geoprocessamento, destacando a importância do monitoramento remoto, da *Internet* das Coisas (IoT), da robótica, da automação agrícola e da inteligência artificial. Por fim, são discutidas as tecnologias de navegação por satélite, fundamentais para o rastreamento e posicionamento em tempo real no campo.

2.1 Agricultura de precisão e geoprocessamento

A agricultura de precisão tem como finalidade aumentar a eficiência das operações agrícolas realizadas no campo, para tal aumento são necessárias às coletas e análise de dados do solo, do clima e a topografia da área que está sendo cultivada. Ou seja, essas tecnologias utilizam ferramentas como: sensores, drones e sistemas de posicionamento (GPS) para realizar monitoramento em tempo real, fornecendo uma aplicação precisa de fertilizantes e insumos agrícolas (ZHANG, 2016). Esse conceito visa a redução do desperdício dos insumos e diminuir os impactos negativos ambientais.

Além disso, o geoprocessamento e os Sistemas de Informação Geográfica (SIG) no setor agrícola vem dando contribuição essencial, desenvolvendo mapas com informações detalhadas das áreas de cultivos e identificando regiões com maior e menor índice de produtividade. Com a utilização dessas ferramentas o planejamento agrícola se torna mais eficiente e sustentável (NASCIMENTO, 2017)

2.2 Monitoramento remoto e *Internet* das Coisas (IoT)

O monitoramento remoto realizado por meio de drones e imagens via satélite auxiliam os produtores a identificar de uma forma mais rápida problemas existentes na lavoura, como pragas, doenças ou estresse hídrico, evitando que causem prejuízos significativos (SALAM,2020). Essa tecnologia vem auxiliando o produtor a tomar decisões mais rápidas e precisa reduzindo perdas e aumentando a produção.

Segundo Salam (2020), a IoT tem apresentado resultados revolucionários na agricultura ao conectar dispositivos inteligentes no campo como: sensores de solo e sistemas de irrigação automatizados. Permitindo que seja realizado o monitoramento contínuo das condições ambientais, proporcionando informações em tempo real a fim que possa ser tomada decisões baseadas em dados.

Com o uso de sensores inteligentes na agricultura para medir umidade, temperatura e nutrientes do solo vem garantindo uma gestão mais eficiente.

2.3 Robótica e automação na agricultura

Com a automação agrícola a dependência de mão de obra sem especialização vem sendo reduzida cada vez mais e aumentando a precisão das operações agrícolas no campo.

Equipamentos como, robôs e tratores estão sendo utilizados nos setores como plantio, colheita e poda, melhorando a eficiência operacional (HUANG, 2021). O desenvolvimento de equipamento agrícolas inteligentes também permite realizações de operações com maior precisão, reduzindo custos e aumentando a produtividade (BOCHTIS, 2020).

2.4 Inteligência artificial

A inteligência artificial (IA) tem sido utilizada na agricultura para analisar grandes quantidades de dados, identificando padrões que possam auxiliar no planejamento da produção.

Fatores preditivos baseados em aprendizado de máquina ajudam a prever o rendimento das safras, a fim de reduzir o uso de insumos e identificar riscos climáticos (ABRAHAM; JAIN, 2021). Além disso, algoritmos inteligentes baseados em IA podem

auxiliar em boas práticas agrícolas tornando as operações mais eficientes, aumentando a produtividade e diminuindo o impacto ambiental.

2.5 GNSS (Sistema de Navegação por Satélite)

A tecnologia de navegação por satélite tem se tornado cada vez mais presente no nosso dia a dia, permitindo determinar com precisão a localização em praticamente qualquer lugar do mundo. Nesse contexto, é importante destacar que o GNSS (*Global Navigation Satellite System*) representa o conjunto de todos os sistemas de navegação por satélite existentes, englobando diferentes constelações que operam de forma independente ao redor do planeta (ALVES, 2013). Entre esses sistemas que compõem o GNSS está o GPS, muito conhecido popularmente, mas que corresponde apenas a uma das constelações que integram esse conjunto mais amplo.

Segundo García Álvarez (2009), o GNSS é formado por sistemas de navegação baseados em satélites como o GPS, o GLONASS e o Galileo, que permitem determinar com precisão a posição espacial e temporal em qualquer ponto da Terra. Embora hoje seja amplamente utilizado, seu desenvolvimento é relativamente recente: teve origem na década de 1970 com o GPS norte-americano, inicialmente criado para fins militares. A partir da década de 1990, a tecnologia passou a ser utilizada em aplicações civis, expandindo seu uso globalmente com o apoio da cooperação internacional.

O GNSS é caracterizado por diversos sistemas independentes, os mais conhecidos são: GPS desenvolvido pelos Estados Unidos, GLONASS pela Rússia, Galileo pela União Europeia e BeiDou da China. Cada um desses sistemas individualmente possui uma constelação de satélites que tem a função de transmitir sinais para receptores localizados na superfície da terra, com isso permitindo o cálculo da posição através da triangulação (ALVES, 2013).

Segundo Alves (2013), os erros de posicionamento no GNSS podem ser causados por alguns fatores, esses erros podem ser causados por: Interferências ionosféricas, multipercurso e erros causados nos relógios atômicos dos satélites. Para diminuir esses erros, utilizam-se técnicas como o Posicionamento por Ponto Preciso (PPP) e o Posicionamento Relativo, que permitem um aumento significativo na precisão dos dados obtidos.

De acordo com García Álvarez (2009), os sistemas de navegação por satélite representam um campo promissor e multidisciplinar, oferecendo amplas oportunidades de atuação para profissionais da área de telecomunicações. Esses sistemas envolvem atividades em órgãos reguladores, empresas de tecnologia e fabricação de hardware, além de agências internacionais como a ESA.

García Álvarez (2009) também ressalta que o desenvolvimento dos sistemas GNSS encontra-se em um momento crucial, com potencial para revolucionar a vida cotidiana de forma semelhante à internet e à telefonia móvel. A implementação do sistema europeu Galileo e a criação de centros de controle especializados marcam o avanço da cooperação internacional e o fortalecimento dessa tecnologia. O autor observa que o setor de sistemas GNSS está se consolidando como uma realidade concreta, gerando novas oportunidades de emprego e formação profissional, e que o domínio prévio dessa área representa uma vantagem competitiva significativa para os profissionais que desejam atuar nesse segmento.

Entre os sistemas que compõem o GNSS, destaca-se o GPS, que é o mais utilizado e será detalhado na seção seguinte.

2.6 GPS (Sistema de Posicionamento Global)

Como apresentado na seção anterior, o GPS (Global Positioning System) é um dos sistemas que compõem o GNSS, sendo a constelação desenvolvida pelos Estados Unidos para operações de posicionamento em escala global. Embora seja o sistema mais conhecido e amplamente utilizado, ele corresponde apenas a uma parte do conjunto maior de sistemas de navegação por satélite.

De acordo com CORREA (2014) conceitua GPS como um sistema de navegação ou localização global altamente avançado, o qual fornece com precisão a latitude, longitude e altitude de uma área/ponto específico. Esse sistema opera em todas as condições climáticas, consistindo em uma constelação de 24 satélites artificiais posicionados em órbitas a aproximadamente 20 mil quilômetros acima da superfície da Terra.

O Sistema de Posicionamento Global (GPS), desenvolvido pelo Departamento de Defesa dos Estados Unidos, é um sistema de radionavegação criado inicialmente para fins militares, com o objetivo de servir como principal meio de navegação das forças armadas. Posteriormente, devido à sua alta precisão e à evolução tecnológica

dos receptores, o uso civil tornou-se predominante, abrangendo áreas como navegação, agricultura, geodésia e transporte (Monico, 2000).

De acordo com Monico (2000), o GPS tem ampliado significativamente as atividades que envolvem posicionamento, oferecendo alta precisão e confiabilidade. Sua integração com outros sistemas de comunicação possibilitou o surgimento de novos conceitos de posicionamento, os chamados sistemas ativos. As aplicações do GPS são diversas e continuam em constante expansão, abrangendo desde a geodinâmica, navegação global e regional, levantamentos geodésicos e controle de deformações, até áreas como agricultura de precisão, estudos atmosféricos, turismo e pesca.

A seguir, é apresentado o protótipo desenvolvido, descrevendo as tecnologias utilizadas, a organização das funcionalidades e a forma como os recursos de geolocalização foram incorporados ao sistema.

3. Protótipo desenvolvido

Este capítulo apresenta as principais etapas que levaram ao desenvolvimento do protótipo, descrevendo as tecnologias utilizadas, os requisitos funcionais estabelecidos e a modelagem do banco de dados. Também são detalhados os *scripts* mais relevantes e a estrutura adotada para cada funcionalidade do sistema. Por fim, é apresentada a *interface* do protótipo, destacando suas principais telas. A seguir, são descritas as tecnologias que serviram de base para o desenvolvimento da aplicação.

3.1 Tecnologias utilizadas

No desenvolvimento do protótipo foi utilizada a linguagem *Dart* (Dart, 2025), empregada pelo ecossistema *Flutter* (Google, 2025) no desenvolvimento de aplicações multiplataforma e conhecida pelo suporte a tipagem estática e programação assíncrona, o que contribui para agilidade e bom desempenho em operações de rede e de sensores como GPS. O projeto foi implementado com *Flutter*, *framework* de código aberto mantido pelo Google, que possibilita criar aplicações para *Android*, *iOS*, *web* e *desktop* a partir de uma única base de código, adotando uma arquitetura baseada em *widgets* para compor interfaces gráficas versáteis e eficientes (Google, 2025). O ambiente também oferece recursos de produtividade como *hot reload*, favorecendo ciclos de interação rápidos durante o desenvolvimento (Google, 2025).

No contexto deste trabalho, o *Flutter* foi utilizado para gerar executáveis para *Windows* (*desktop*) e *Android* (*mobile*), mantendo consistência visual e funcional entre as plataformas. Em termos conceituais, o *Flutter* pode ser entendido como um *toolkit* de UI (*User Interface*) multiplataforma que produz aplicativos nativamente compilados a partir de um único código-fonte, promovendo alta reutilização e desempenho consistente entre diferentes sistemas (Google, 2025). Já o *Dart*, por ser a linguagem nativa do *Flutter*, fornece os mecanismos necessários para lidar com concorrência e I/O (*Input/Output*) sem bloquear a interface, favorecendo a experiência do usuário em cenários de coleta de dados em campo e sincronização com o *backend* (Dart, 2025).

Para armazenamento utilizou-se para desenvolvimento deste protótipo o *Supabase*, uma plataforma hospedada em nuvem que provisiona um *backend*

completo baseado em *PostgreSQL*, (SUPABASE, 2025).

O *PostgreSQL* é gerenciado no *Supabase*, viabilizando um banco de dados relacional em nuvem com implantação simples e baixo custo operacional. O esquema inclui tabelas de cadastros, tela de abertura de apontamentos para inserção de dados e relacionamentos por chaves estrangeiras, atendendo às necessidades de registro e consulta do sistema

3.2 Requisitos funcionais

Os requisitos funcionais a seguir explicam, de forma direta, o que o sistema precisa fazer para cumprir os objetivos do projeto. Eles cobrem do registro de apontamentos em campo à gestão de cadastros, passando pela vinculação das coletas às ordens de serviço. Para cada item, indicamos quem usa, quais dados entram, quais regras valem e o que sai. Além dos critérios de aceitação, que ajudam a verificar, na prática, se a funcionalidade está correta. A tabela resumida organiza tudo isso de um jeito fácil de consultar e manter, conectado às telas do aplicativo (*Windows/Android*) às tabelas do banco de dados (*PostgreSQL/Supabase*).

Na **Tabela 1** são apresentados os requisitos funcionais para a inserção de um registro de apontamento.

Tabela 1 - Representação requisito funcionais registro de apontamento.

Descrição	Atores	Entradas	Regras validações	Saída efeitos	Critérios de aceitação	Telas relacionadas
Permitir registrar um apontamento vinculando: operador, equipamento e ordem de serviço.	Operador	Matrícula do operador, Nº do equipamento Nº da ordem de serviço. .	Operador, equipamento e ordem de serviço devem estar cadastrados.	Inserção de um registro apontamento com nº de boletim gerado	Com os dados válidos, grava o registro.	Telas: Abertura de apontamento e Boletim de apontamento

FONTE: elaborada pelo autor.

Na **Tabela 2** são apresentados os requisitos funcionais relacionados ao processo de inserção de um registro de apontamento de pontos GPS. A tabela detalha os elementos essenciais para a operação, incluindo a descrição da funcionalidade, os

atores envolvidos, os dados de entrada necessários, as regras de validação, os efeitos gerados após o processamento, os critérios de aceitação e as telas vinculadas ao procedimento. Essa estruturação permite compreender de forma clara como o sistema deve se comportar ao registrar leituras de latitude e longitude associadas a um boletim.

Tabela 2 - Representação requisito funcionais registro de pontos GPS.

Descrição	Atores	Entradas	Regras validações	Saída efeitos	Critérios de aceitação	Telas relacionadas
Registrar leituras de latitude e longitude e associar a um boletim.	Operador	Nº do boletim, latitude e longitude.	Nº do boletim deve existir. Latitude e longitude são dados de entrada obrigatórias.	Inserção de dados de latitude e longitude associado ao Nº de boletim.	Com boletim válido, grava o ponto. Com boletim inexistente, recusa e informa.	Telas:Boletim de apontamento e rastreamento por GPS.

FONTE: elaborada pelo autor.

Na **Tabela 3** são apresentados os requisitos funcionais relacionados ao processo de vincular um boletim a uma ordem de serviço (OS). A tabela descreve os dados necessários para essa operação, como número do boletim, número da OS e informações de horímetro, além das validações que garantem a consistência desses registros. Esse requisito assegura que cada atividade mecanizada seja corretamente associada à sua respectiva ordem de serviço, permitindo maior controle e organização das operações realizadas em campo.

Tabela 3 - Representação requisito funcionais vincular ordem de serviço (OS) ao boletim de atividades mecanizada.

Descrição	Atores	Entradas	Regras validações	Saída efeitos	Critérios de aceitação	Telas relacionadas
Vincular um boletim a uma ordem de serviço.	Operador	Nº do boletim e Nº da ordem de serviço. Horímetro inicial, horímetro final e total	Nº de boletim e número de OS deve existir. Horímetro final deve ser maior que inicial	Inserção de registro com informações da ordem de serviço vinculado ao boletim	Com os dados de entrada validados, gravar registro no banco de dados	Tela: Boletim de apontamento

FONTE: elaborada pelo autor.

Na **Tabela 4** são apresentados os requisitos funcionais relacionados às operações de inserção, consulta e alteração de cadastros. Esses requisitos descrevem os campos necessários, as validações aplicadas e os efeitos esperados em cada ação realizada pelo administrador. Esse conjunto de regras garante que os registros sejam manipulados de forma consistente e segura, assegurando a integridade dos dados e o bom funcionamento do módulo de cadastros do sistema.

Tabela 4 - Requisitos funcionais do módulo de cadastros (inserção, consulta e alteração).

Descrição	Atores	Entradas	Regras validações	Saída efeitos	Critérios de aceitação	Telas relacionadas
*Inserir. *Alterar. *Consultar.	Administrador	Campos de cada entidade.	Campos obrigatórios não podem estar vazios ou nulos. Exemplo: chaves primárias.	registros consultados, novos registros inseridos ou alterados.	Com os dados válidos realizar: consultas, inserção e alteração dos registros.	Telas: Todas as telas referente ao módulo de cadastro do sistema.

FONTE: elaborada pelo autor.

Na **Tabela 5** são apresentados os requisitos funcionalmente associados às operações de consulta e filtragem de registros, que permitem ao usuário analisar os dados inseridos no sistema e gerar relatórios conforme os períodos e critérios informados. Esses requisitos definem os dados de entrada, as validações necessárias e os resultados esperados, assegurando que o processo de consulta seja realizado de forma eficiente e consistente.

Tabela 5 - Representação requisito funcionais consultar/filtra registro gerar relatórios.

Descrição	Atores	Entradas	Regras validações	Saída efeitos	Critérios de aceitação	Telas relacionadas
Realizar consultas, filtros e gerar relatórios.	Administrador	Inserir dados para serem filtrados e informar períodos (datas).	Dados sendo filtrados sejam válidos.	listas, Relatórios gerados.	Com período e dados válidos retornam: listas e relatórios.	Tela: Relatórios.

FONTE: elaborada pelo autor.

Na **Tabela 6** são apresentados os requisitos funcionais referentes ao processo de validação e autenticação de usuário. A tabela descreve os dados utilizados na verificação de acesso, as regras aplicadas para confirmar as credenciais e os efeitos resultantes após a autenticação. Esses requisitos garantem que apenas usuários devidamente cadastrados possam acessar o sistema, mantendo a segurança e a integridade das informações.

Tabela 6 - Representação requisito funcionais validação/autenticação de usuário.

Descrição	Atores	Entradas	Regras validações	Saída efeitos	Critérios de aceitação	Telas relacionadas
Autenticar usuário user_name senha_hash	Todos	user_name e senha	Verificar: usuário cadastrado e senha_hash com bcrypt	Acesso às telas conforme o perfil do usuário.	Credenciais válidas: acesso concedido. Inválidas, recusa e permanece na tela de login.	Tela:Acesso login.

FONTE: elaborada pelo autor.

A partir dos requisitos funcionais apresentados, ficou definido de forma objetiva o que o protótipo deve executar para sustentar a coleta de dados em campo e a gestão de cadastros e abertura de ordens de serviço no *desktop*. Esses requisitos orientam o desenvolvimento, os testes e a validação, garantindo relacionamento entre as telas do protótipo e as entidades do banco de dados.

Na próxima seção será discutido modelagem do banco de dados e suas entidades e cardinalidades e relacionamentos entre entidades.

3.3 Modelagem e estrutura do banco de dados

Serão apresentados a modelagem e a estrutura do banco de dados relacional que sustenta o protótipo. São descritas as entidades, seus atributos, e os relacionamentos que garantem integridade e coerência das informações, bem como decisões de projeto adotadas para atender aos requisitos funcionais.

3.3.1 Entidades e seus atributos

O esquema foi concebido para separar cadastro (fazenda, talhão, operador,

equipamento, operação centro de custo, usuário e login) de registro operacionais (apontamento, apontamento ordem de serviço e pontos GPS) permitindo relacionamento entre coletas de dados no campo e abertura de ordem de serviço. As chaves estrangeiras asseguram ciclos consistentes entre tabelas.

Na **Figura 3.1** é representada a entidade fazenda com atributos; **cd_fazenda**, **des_fazenda** e **data_cadastro**. Sendo que o atributo **cd_fazenda** é chave primária da entidade.

Figura 3.1. Visualização do esquema da entidade fazenda. FONTE: elaborada pelo autor.

Na **Figura 3.2** representa a entidade talhao com atributos; **cd_talhao**, **cd_fazenda**, **des_talhao**, **area_talhao** e **data_cadastro**. Sendo que o atributo **cd_talhao** é chave primária da entidade. O atributo **cd_fazenda** é chave estrangeira que faz referência a entidade fazenda.

Figura 3.2. Visualização do esquema da entidade talhao. FONTE: elaborada pelo autor.

Na **Figura 3.3** representa a entidade operador com atributos; **cd_operador**, **des_operador**, **des_cargo** e **data_cadastro**. Sendo que o atributo **cd_operador** é chave primária da entidade.

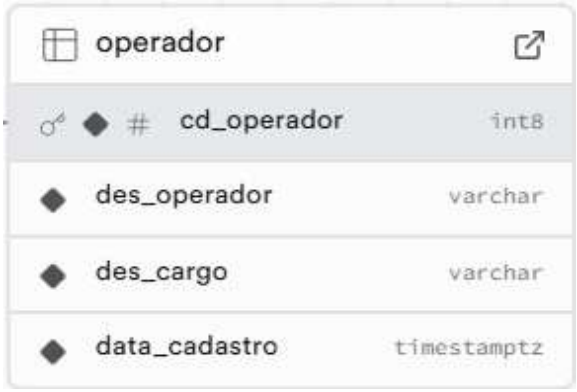

Figura 3.3. Visualização do esquema da entidade operador. FONTE: elaborada pelo autor.

Na **Figura 3.4** representa a entidade equipamentos com atributos; **cd_equipamento**, **des_equipamento**, **des_placa_numero_serie**, **des_fabricante** e **data_cadastro**. Sendo que o atributo **cd_equipamento** é chave primária da entidade.

Figura 3.4. Visualização do esquema da entidade equipamentos. FONTE: elaborada pelo autor.

Na **Figura 3.5** representa a entidade operacao com atributos; **cd_operacao**, **des_operacao** e **data_cadastro**. Sendo que o atributo **cd_operacao** é chave primária da entidade.

Figura 3.5. Visualização do esquema da entidade operacao. FONTE: elaborada pelo autor.

Na **Figura 3.6** representa a entidade centrocusto com atributos; **cd_custo**, **des_custo** e **data_cadastro**. Sendo que o atributo **cd_custo** é chave primária da entidade.

Figura 3.6. Visualização do esquema da entidade centrocusto. FONTE: elaborada pelo autor.

Na **Figura 3.7** representa a entidade usuario com atributos; **cd_usuario**, **des_usuario** e **data_cadastro**. Sendo que o atributo **cd_usuario** é chave primária da entidade.

Figura 3.7. Visualização do esquema da entidade usuario. FONTE: elaborada pelo autor.

Na **Figura 3.8** representa a entidade *login* com atributos; **cd_user_name**, **user_name**, **senha_*hash***, **data_cadastro** e **cd_usuario**. Sendo que o atributo **cd_user_name** é chave primária da entidade. O atributo **cd_usuario** é uma chave estrangeira que faz referência a entidade **usuario**.

Figura 3.8. Visualização do esquema da entidade *login*. FONTE: elaborada pelo autor.

Na **Figura 3.9** representa a entidade **ordem_servico** com atributos; **cd_ordem_servico**, **data_cadastro**, **data_fechamento**, **cd_operacao**, **cd_custo**, **cd_fazenda**, ***status*** e **talhao**. Sendo que o atributo **cd_ordem_servico** é chave primária da entidade. O atributo **cd_operacao** é uma chave estrangeira que faz referência a entidade **operacao**. Atributo **cd_custo** é chave estrangeira que faz referência a entidade **centrocusto**. Atributo **cd_fazenda** é chave estrangeira que faz referência a entidade **fazenda**.

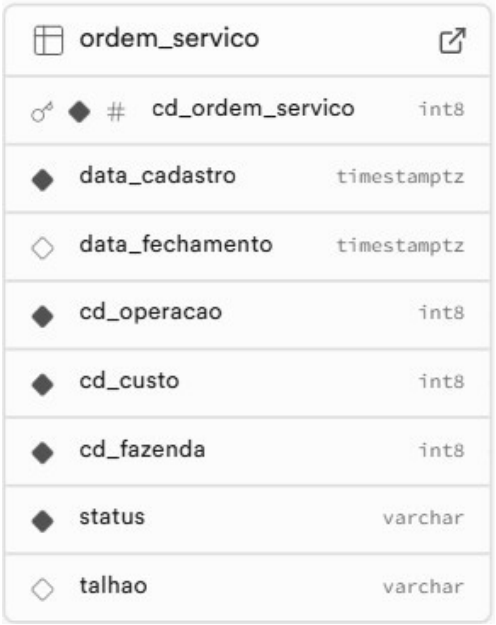

Figura 3.9. Visualização do esquema da entidade ordem_servico. FONTE: elaborada pelo autor.

Na **Figura 3.10** representa a entidade apontamento com atributos; **cd_ boletim**, **cd_operador**, **cd_equipamento**, **turno_h_inicial**, **turno_h_final**, e **data_cadastro**. Sendo que o atributo **cd_boletim** é a chave primária da entidade. O atributo **cd_operador** é uma chave estrangeira que faz referência a entidade

operador. Atributo **cd_equipamento** é chave estrangeira que faz referência a entidade **equipamentos**.

Figura 3.10. Visualização do esquema da entidade apontamento. FONTE: elaborada pelo autor.

Na **Figura 3.11** representa a entidade **pontos_gps** com atributos; **cd_pontos_gps**, **latitude**, **longitude**, **cd_boletim**. Sendo que o atributo **cd_pontos_gps** é a chave primária da entidade. O atributo **cd_boletim** é uma chave estrangeira que faz referência a entidade **apontamento**.

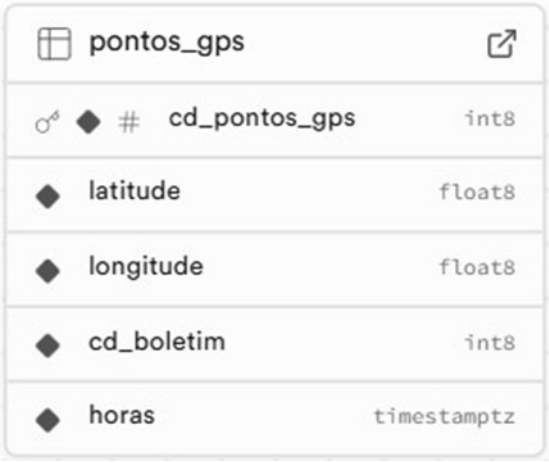

Figura 3.11. Visualização do esquema da entidade pontos_gps. FONTE: elaborada pelo autor.

Na **Figura 3.12** representa a entidade **apontamento_os** com atributos; **cd_apontamento_os**, **cd_boletim**, **cd_ordem_servico**, **horimetro_inicial**, **horimetro_final** e **total**. Sendo que o atributo **cd_apontamento_os** é a chave primária da entidade. O atributo **cd_boletim** é uma chave estrangeira que faz referência a entidade **apontamento**. Atributo **cd_ordem_servico** é uma chave

estrangeira que faz referência a entidade **ordem_servico**.

Figura 3.12. Visualização do esquema da entidade apontamento_os. FONTE: elaborada pelo autor.

Com a modelagem concluída, cada entidade é descrita com seus atributos e com suas chaves primárias e estrangeiras. Na sequência, serão discutidas as cardinalidades presentes no projeto, além de diagramas que evidenciam as relações entre elas.

3.3.2 Cardinalidades e relacionamentos entre entidades

No projeto, relacionamento é o vínculo lógico entre duas entidades (tabelas) por meio de chaves estrangeiras, enquanto cardinalidade indica quantos registros de uma entidade podem associar-se a registros de outra (ex.: 1:1, 1:N, N:M).

Segundo Elmasri e Navathe (2011), em um banco de dados relacional, a integridade referencial assegura a consistência entre tabelas por meio de chaves estrangeiras: todo valor que referência outra relação deve corresponder a uma tupla existente (ou ser nulo quando permitido). Assim, atributos que funcionam como chaves estrangeiras — como o departamento de um funcionário — precisam combinar com a chave primária da tabela referenciada, garantindo que os vínculos entre tuplas permaneçam válidos.

No modelo Entidade-Relacionamento, a cardinalidade define o mapeamento máximo entre entidades (1:1, 1:N, N:M) e a participação indica o mínimo exigido (total ou parcial), caracterizando as restrições estruturais dos relacionamentos (ELMASRI;

NAVATHE, 2011).

Com base nesses conceitos, no projeto adota-se o relacionamento entre as entidades **fazenda** e **talhao**, em que uma fazenda pode conter vários **talhões**, enquanto cada **talhão** pertence a uma única **fazenda**. A **Figura 3.13** ilustra essa associação por meio da chave estrangeira **talhao.cd_fazenda** e explicita a cardinalidade **1:N (Fazenda → Talhão)**, utilizada para garantir relacionamento e integridade entre os registros.

Figura 3.13. Relacionamento fazenda–talhao com cardinalidade 1:N via talhao.cd_fazenda. FONTE: elaborada pelo autor.

Na **Figura 3.14** representa o relacionamento entre **operador** e **apontamento** é do tipo 1:N. Um mesmo operador pode estar associado a vários **apontamentos**, enquanto cada **apontamento** referência um único **operador**. A ligação ocorre por meio da chave estrangeira **apontamento.cd_operador**, que referência **operador.cd_operador**, garantindo a integridade referencial entre as entidades.

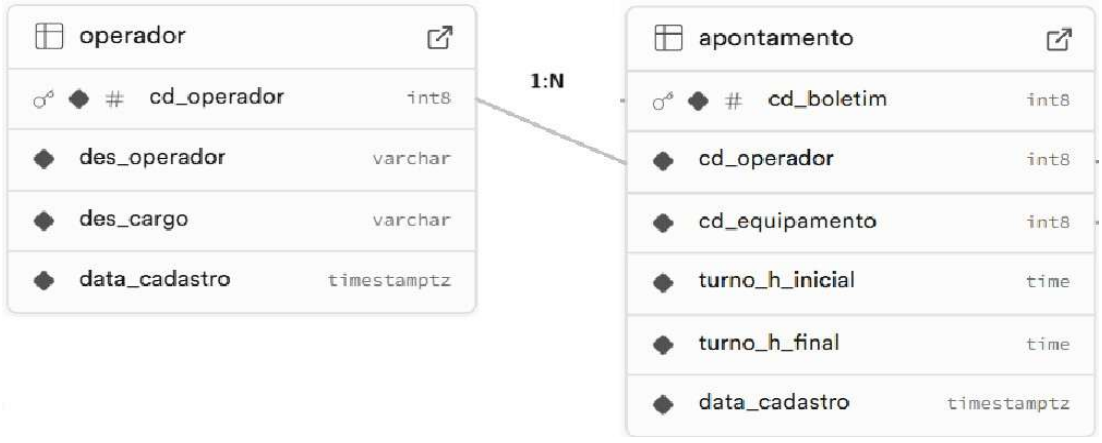

Figura 3.14. Relacionamento operador–apontamento com cardinalidade 1:N via apontamento.cd_operador. FONTE: elaborada pelo autor.

Na **Figura 3.15** representa o relacionamento entre **equipamento** e **apontamento** do tipo 1:N. Um mesmo **equipamento** pode estar associado a vários **apontamentos**, enquanto cada **apontamento** referência um único **equipamento**. A ligação ocorre por meio da chave estrangeira **apontamento.cd_equipamento**, que referência **equipamentos.cd_equipamento**, garantindo a integridade referencial e o relacionamento adequado entre as tabelas.

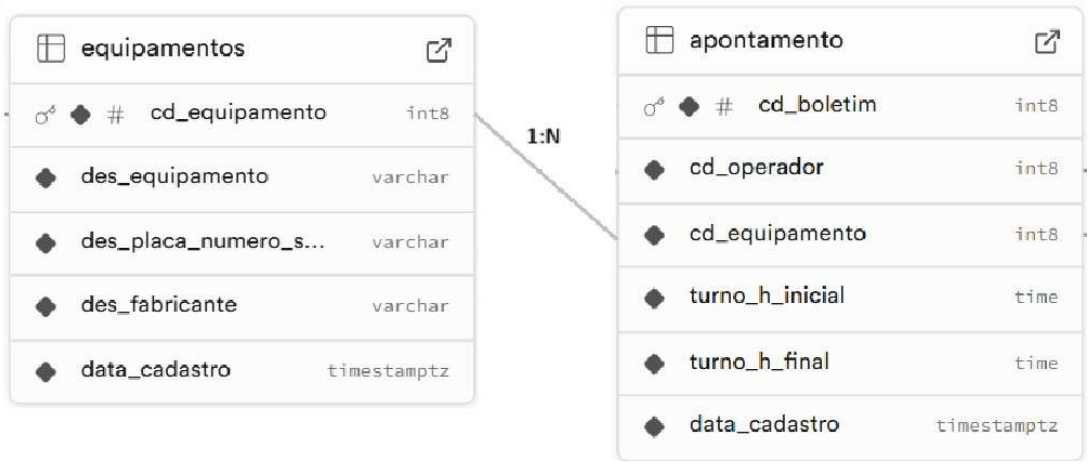

Figura 3.15. Relacionamento equipamentos–apontamento cardinalidade 1:N apontamento.cd_equipamento. FONTE: elaborada pelo autor.

Na **Figura 3.16** representa o relacionamento entre **apontamento** e **pontos_GPS** é 1:N. Um mesmo **apontamento** pode agrupar várias leituras de GPS, enquanto cada leitura pertence a um único **apontamento**. A vinculação ocorre pela chave estrangeira **pontos_gps.cd_boletim**, referência **apontamento.cd_boletim**, garantindo integridade referencial e permitindo reconstruir a trilha de localização de cada boletim.

Figura 3.16. Relacionamento apontamento–pontos_gps com cardinalidade 1:N via pontos_gps.cd_boletim. FONTE: elaborada pelo autor.

Na **Figura 3.17** referente ao relacionamento entre **apontamento** e **apontamento_OS** é 1:N. Um mesmo apontamento (boletim) pode ter vários vínculos em **apontamento_os**, enquanto cada linha de **apontamento_os** pertence a um único boletim. A ligação é feita pela chave estrangeira **apontamento_os.cd_boletim** que referência a entidade **apontamento**, garantindo integridade referencial e permitindo registrar **horímetro inicial/final** e o **total**.

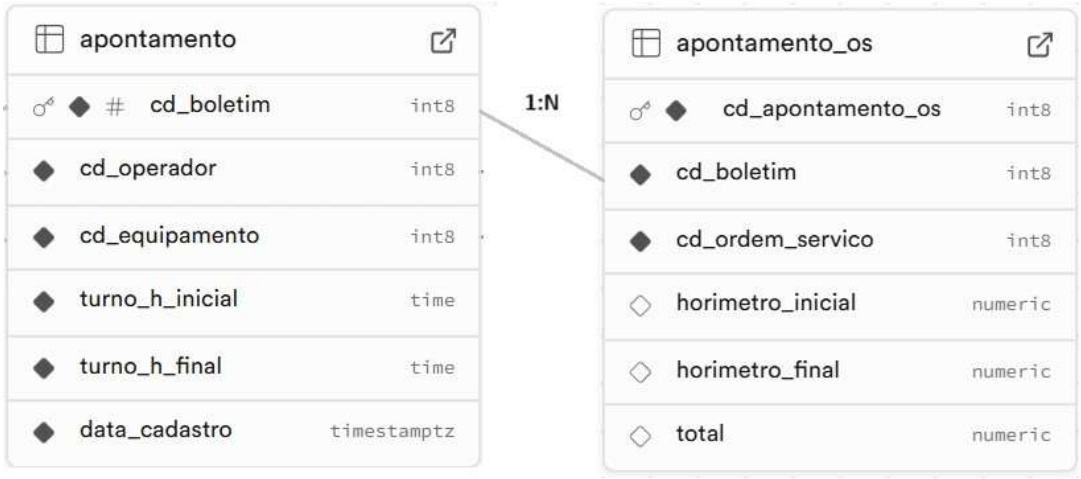

Figura 3.17. Relacionamento apontamento–apontamento_os com cardinalidade 1:N via apontamento_os.cd_boletim. FONTE: elaborada pelo autor.

Na **Figura 3.18** apresenta o relacionamento entre **operacao**, **centrocusto**, **fazenda** e **ordem_servico**. A entidade **ordem_servico** se vincula, com cardinalidade 1:N, aos cadastros **operacao** (**ordem_servico.cd_operacao** faz referência a **operacao.cd_operacao**), **centrocusto** (**ordem_servico.cd_custo** faz referência a centrocusto.cd_custo) e **fazenda** (**ordem_servico.cd_fazenda** faz referência a **fazenda.cd_fazenda**). Em todos os vínculos, a participação é total do lado de **ordem_servico** (campos obrigatórios) e parcial do lado dos cadastros, o que assegura a integridade referencial e viabiliza consultas e relatórios por tipo de atividade, alocação de custos e propriedade.

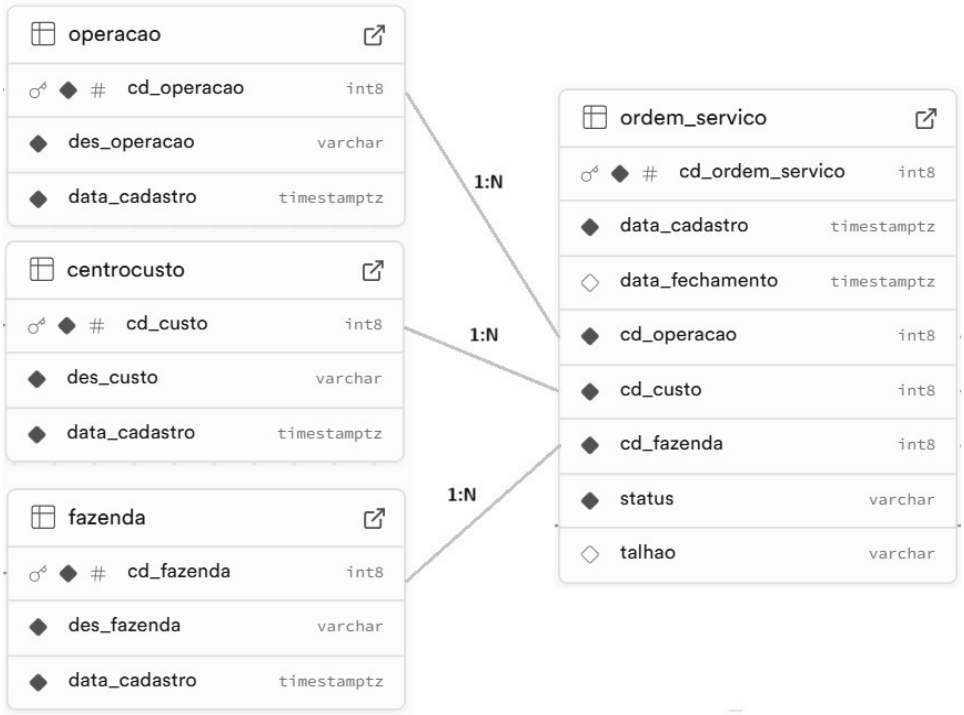

Figura 3.18. Relacionamento operacao, centrocusto e fazenda relacionados a ordem_Serviço (cardinalidade 1:N). FONTE: elaborada pelo autor.

Na **Figura 3.19** apresenta o relacionamento entre **ordem_servico** e **apontamento_os** é 1:N. Uma mesma ordem de serviço pode possuir vários vínculos em **apontamento_os**, enquanto cada registro de **apontamento_os** referência uma única ordem de serviço. A ligação é feita pela chave estrangeira **apontamento_os.cd_ordem_servico** (referência a **ordem_servico.cd_ordem_servico**), o que assegura a integridade referencial e permite registrar **horímetro inicial/final** e o **total** associado.

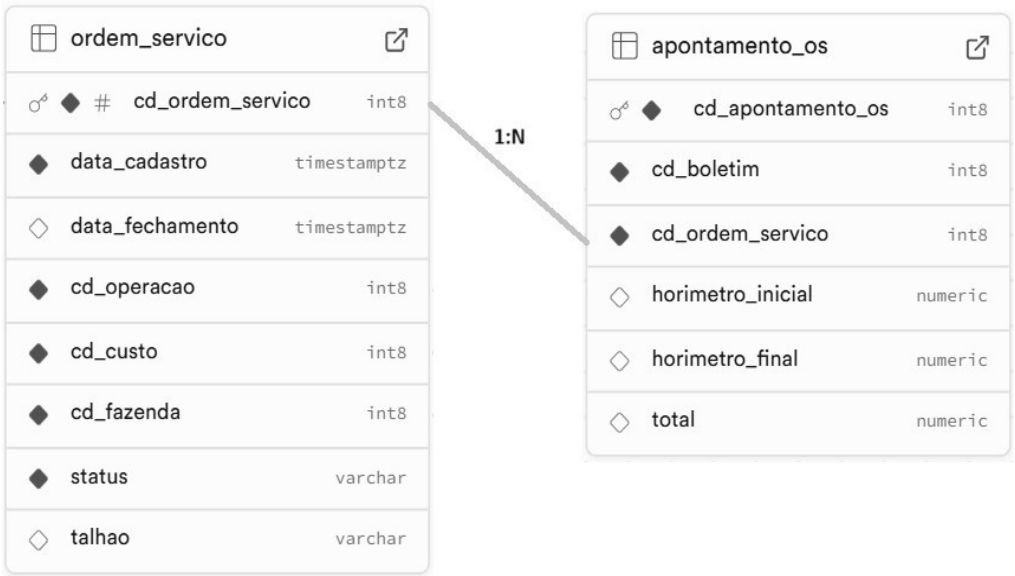

Figura 3.19. Relacionamento ordem_serviço–apontamento_os (1:N) via apontamento_os.cd_ordem_servico. FONTE: elaborada pelo autor.

Na **Figura 3.20** apresenta o relacionamento entre usuario e *login* é 1:N. Um mesmo usuário pode possuir várias credenciais (l*ogins*), enquanto cada registro em *login* está vinculado a um único usuário. A ligação é feita pela chave estrangeira ***login*.cd_usuario**, que referência **usuario.cd_usuario**, garantindo a integridade referencial entre as tabelas.

Figura 3.20. Relacionamento usuario–login (1:N) via login.cd_usuario. FONTE: elaborada pelo autor.

Na **Figura 3.21** apresenta a tabela **apontamento_os**, que liga **apontamento** e **ordem_servico**. Um boletim pode ter várias linhas em **apontamento_os** (1:N) e cada linha pertence a uma única ordem de serviço (N:1). Esse conceito já foi ilustrado nas **Figuras 3.17** e **3.19**. Em conjunto, esses dois vínculos 1:N materializam o relacionamento N:M entre **apontamento** e **ordem_servico**, permitindo associar um boletim a várias ordens de serviço e, ao mesmo tempo, uma ordem de serviço a vários **boletins**, com integridade e relacionamento entre as entidades.

Figura 3.21. Relacionamento N:M entre apontamento e ordem_servico mediada por apontamento_os. FONTE: elaborada pelo autor.

Foram apresentados os principais relacionamentos e suas cardinalidades no projeto, evidenciando como as tabelas se ligam por chaves estrangeiras e como isso garante integridade, relacionamento e suporte às consultas e relatórios. Os exemplos mostraram vínculos 1:N predominantes e a relação N:M mediada por apontamento_os entre apontamento e ordem_servico.

Na sequência, a próxima seção aborda as *views* do projeto, descrevendo sua finalidade, estrutura e como elas simplificam consultas e relatórios.

3.3.3 Visões (*views*) do projeto

View (visão) é uma tabela virtual definida por uma consulta (*structured query language* (SQL) sobre uma ou mais tabelas. Ela não armazena dados próprios; exibe resultados calculados no momento da consulta. Serve para simplificar relatórios, padronizar consultas e restringir o acesso a colunas/linhas (ELMASRI; NAVATHE, 2011).

Segundo Elmasri e Navathe (2011), uma *view* (visão) é uma “tabela virtual” definida por uma consulta sobre uma ou mais tabelas base. Ela não precisa existir fisicamente com dados próprios: o sistema de gerenciamento de banco de dados (SGBD) armazena apenas a definição e, ao ser consultada, recalcula o resultado como se fosse uma tabela comum. As *views* podem ter restrições para atualização (inserções/alterações/remoções), mas não impõem limitações à consulta dos dados; além disso, podem ser definidas tanto a partir de tabelas quanto de outras *views*.

A especificação de uma view em *SQL* é feita com o comando *CREATE VIEW*, no qual se define um nome para a tabela virtual, uma lista de atributos quando necessário e a consulta que determina seu conteúdo. Se os atributos resultam diretamente das tabelas de base, sem funções ou operações que alterem nomes, não é preciso renomeá-los, pois eles herdam os nomes das colunas de origem por padrão (ELMASRI; NAVATHE, 2011).

Com base nesses conceitos, o projeto adota a *view* **vw_boletim_apontamento_mecanizado**, concebida para unificar, em uma única consulta, as informações essenciais do boletim: responsável (operador), máquina (equipamento), ordem de serviço, operação, centro de custo, fazenda/talhão, horários do turno e total de horímetro. Com isso, o administrador passa a dispor de uma fonte padronizada e consistente para elaborar relatórios operacionais por período, sem

repetir *joins*[1] ou regras de consolidação no aplicativo.

Além de simplificar a consulta, a view permite relatórios direcionados apenas com *SELECT*[2] e filtros que combinam a data do apontamento com diferentes dimensões do negócio. Exemplos de entradas usuais incluem: (**apontamento_data_cadastro**, **des_operador**), (**apontamento_data_cadastro**, **cd_equipamento**), (**apontamento_data_cadastro**, **cd_ordem_servico**), (**apontamento_data_cadastro**, **des_operacao**) e (**apontamento_data_cadastro**, **des_fazenda**). Dessa forma, é possível obter, com rapidez e precisão, listagens por dia e por operador, por dia e por máquina, por dia e por OS, ou ainda por dia e por tipo de operação ou fazenda, gerando relatórios mais específicos e úteis à gestão, sem necessidade de *scripts* complexos.

Na **Figura 3.22** demonstra as especificações da *view* **vw_boletim_apontamento_mecanizado** mostrando os campos expostos (**apontamento_data_cadastro**, **cd_boletim**, **des_operador**, **cd_equipamento**, **des_equipamento**, **turno_h_inicial**, **turno_h_final**, **cd_ordem_servico**, **des_operacao**, **des_custo**, **des_fazenda**, **talhao** e **total**), e as *base/joins* (**apontamento**, **operador**, **equipamentos**, **apontamento_os**, **ordem_servico**, **operacao**, **centrocusto** e **fazenda**).

```
1  CREATE OR REPLACE VIEW public.vw_boletim_apontamento_mecanizado AS
2  SELECT
3    a.data_cadastro        AS apontamento_data_cadastro,
4    a.cd_boletim,
5    o.des_operador,
6    e.cd_equipamento,
7    e.des_equipamento,
8    a.turno_h_inicial,
9    a.turno_h_final,
10   os.cd_ordem_servico,
11   op.des_operacao,
12   cc.des_custo,
13   f.des_fazenda,
14   os.talhao,
15   aos.total
16 FROM public.apontamento      AS a
17 JOIN public.operador         AS o  ON o.cd_operador     = a.cd_operador
18 JOIN public.equipamentos     AS e  ON e.cd_equipamento  = a.cd_equipamento
19 JOIN public.apontamento_os   AS aos ON aos.cd_boletim   = a.cd_boletim
20 JOIN public.ordem_servico    AS os  ON os.cd_ordem_servico = aos.cd_ordem_servico
21 JOIN public.operacao         AS op  ON op.cd_operacao   = os.cd_operacao
22 JOIN public.centrocusto      AS cc  ON cc.cd_custo      = os.cd_custo
23 JOIN public.fazenda          AS f   ON f.cd_fazenda     = os.cd_fazenda;
```

Figura 3.22. *Script SQL* da v*iew*: vw_boletim_apontamento_mecanizado. FONTE: elaborada pelo autor.

[1] **JOIN:** operação utilizada em bancos de dados relacionais para combinar registros de duas ou mais tabelas com base em uma coluna relacionada, permitindo recuperar dados correlacionados em uma única consulta.

[2] **SELECT:** comando usado para buscar e exibir dados armazenados em uma tabela do banco de dados.

A segunda *view* do projeto, **vw_boletim_apontamento_gps**, foi criada para apoiar relatórios de área e a verificação de conformidade. Ela consolida, por boletim, a ordem de serviço associada (com **fazenda**, **talhão** e **operação**) e os pontos de GPS coletados durante a execução.

Com essa visão, o administrador consegue checar se a ordem de serviço aberta foi executada na área correta, cruzando data, boletim, ordem de serviço, fazenda, operação, talhão e as coordenadas latitude/longitude registradas em campo.

A **Figura 3.23** apresenta as especificações da *view* **vw_boletim_apontamento_gps**, indicando os campos expostos: **apontamento_data_cadastro**, **cd_boletim**, **cd_ordem_servico**, **des_custo**, **des_fazenda**, **des_operacao**, **talhao**, **latitude** e **longitude**; e as bases e junções utilizadas: **apontamento**, **apontamento_os**, **ordem_servico**, **centrocusto**, **fazenda**, **operacao** e **pontos_gps**.

```
1  CREATE OR REPLACE VIEW public.vw_boletim_apontamento_gps AS
2  SELECT
3    a.cd_boletim,
4    a.data_cadastro        AS apontamento_data_cadastro,
5    aos.cd_ordem_servico,
6    cc.des_custo,
7    f.des_fazenda,
8    op.des_operacao,
9    os.talhao,
10   pg.latitude,
11   pg.longitude
12 FROM public.apontamento      AS a
13 JOIN public.apontamento_os   AS aos ON aos.cd_boletim       = a.cd_boletim
14 JOIN public.ordem_servico    AS os  ON os.cd_ordem_servico  = aos.cd_ordem_servico
15 JOIN public.centrocusto      AS cc  ON cc.cd_custo          = os.cd_custo
16 JOIN public.fazenda          AS f   ON f.cd_fazenda         = os.cd_fazenda
17 JOIN public.operacao         AS op  ON op.cd_operacao       = os.cd_operacao
18 JOIN public.pontos_gps       AS pg  ON pg.cd_boletim        = a.cd_boletim;
```

Figura 3.23. *Script SQL* da v*iew*: vw_boletim_apontamento_gps. FONTE: elaborada pelo autor.

3.4 *Scripts* de programação

A programação foi desenvolvida em *Flutter/Dart* para o cliente móvel e *desktop*, utilizando o *Flutter SDK* (*VS Code/Android Studio*) para compilar e distribuir o aplicativo para *Android* e *Windows*. A integração com o *backend* foi feita por meio do *SDK* do *Supabase*, enquanto os *scripts SQL* (*PostgreSQL*) responsáveis por tabelas, relacionamentos e visões foram executados diretamente no *Supabase*.

3.4.1 Código-fonte

O código foi estruturado para integrar a captura de dados operacionais (cadastros, abertura de ordens de serviço e apontamentos) e de trilhas de GPS no aplicativo, realizar validações e tratamentos no cliente e sincronizar os registros com o *backend* em nuvem via *SDK* do *Supabase* (*PostgreSQL*).

3.4.2 Código carregamento de variáveis de ambiente e inicialização do *Supabase*

A **Figura 3.24** apresenta o trecho do código que realiza o *bootstrap* do serviço de dados. Primeiramente, as variáveis de ambiente são carregadas a partir do arquivo. env, tornando disponíveis, em tempo de execução, a URL do projeto e a chave pública do *Supabase*. Em seguida, os valores são lidos e validados; caso estejam ausentes ou vazios, a inicialização é interrompida com exceção, adotando a estratégia *fail-fast*. Por fim, o *SDK* do *Supabase* é configurado com os parâmetros verificados, concluindo a preparação do *backend* no cliente antes da renderização da *interface*.

```
  /* Carrega as variáveis de ambiente do arquivo .env para a memória do app.
  O await garante que o carregamento termine antes de usar os valores. */
  await dotenv.load(fileName: ".env");

  /* Lê a variável SUPABASE_URL carregada do .env e guarda em url. final indica
  atribuição única (não será reatribuída) */
  /* Lê a chave pública do projeto Supabase. Também é final pelo mesmo motivo. */
  final url = dotenv.env['SUPABASE_URL'];
  final anonKey = dotenv.env['SUPABASE_ANON_KEY'];

  /* Validação defensiva: se url ou anonKey estiverem nulas ou vazias, interrompe a inicialização. */
  if (url == null || url.isEmpty || anonKey == null || anonKey.isEmpty) {
    throw Exception(
      "SUPABASE_URL ou SUPABASE_ANON_KEY não definidos no .env",
    );
  }
  /* Conclui o bootstrap do cliente de dados, inicializando
  o SDK do Supabase com os parâmetros validados. */
  await Supabase.initialize(
    url: url,
    anonKey: anonKey,
);
```

Figura 3.24. Inicialização do cliente *Supabase* com variáveis de ambiente. FONTE: elaborada pelo autor.

3.4.3 Código de autenticação e checagem de credenciais com *hash*

A **Figura 3.25** apresenta trechos de código da tela de *login*. Antes de acionar o *backend*, realiza-se uma pré-validação dos campos: o sistema lê *userName* =

loginAcesso.text.trim() e senha = senhaAcesso.text e interrompe o fluxo se algum estiver vazio, evitando chamadas desnecessárias.

Em seguida, é feita a busca do usuário no *Supabase*, na tabela *login*, selecionando apenas as colunas necessárias e filtrando pela coluna *user_name*. Depois, o sistema recupera o valor de senha_*hash* e verifica se está presente conforme a regra de negócio que exige preenchimento. Por fim, a verificação da credencial ocorre localmente com *BCrypt.checkpw*(senha, *hash*). Se a comparação falhar, o acesso é negado e o usuário recebe uma mensagem informativa. Esse fluxo evita armazenar senhas em texto plano, garante validação no cliente e mantém as credenciais centralizadas no banco.

```dart
Future<void> _login() async {
  // Pré-validação do login: evita chamada desnecessária ao backend.
  // Leitura dos campos
  final userName = loginAcesso.text.trim();
  final senha = senhaAcesso.text;
  // Verifica se algum campo está vazio
  if (userName.isEmpty || senha.isEmpty) {
    _showMsg("Informe usuário e senha.");
    return;
```

```dart
    final Map<String, dynamic>? registroUsuario = await Supabase.instance.client
        .from('login')
        // Selecionando apenas as colunas necessárias
        .select('user_name,senha_hash,cd_usuario')
        // Filtro por user_name
        .eq('user_name', userName)
        // Retorna no máx. 1 linha; se não achar, vem null; se houver >1, lança erro.
        .maybeSingle();
```

```dart
    // Verifica hash (bcrypt)
    final String? hash = registroUsuario['senha_hash'] as String?;
    // Checagem defensiva: evita chamar o bcrypt com valor nulo/vazio
    if (hash == null || hash.isEmpty) {
      _showMsg("Usuário sem senha configurada.");
      return;
    }
    // Comparação segura usando bcrypt
    final ok = BCrypt.checkpw(senha, hash);
    if (!ok) {
      _showMsg("Usuário ou senha incorretos.");
      return;
```

Figura 3.25. Código do fluxo de *login*: pré-validação, consulta ao *Supabase* e verificação com *bcrypt*.
FONTE: elaborada pelo autor.

3.4.4 Código de *insert*, *update* e *select* na tabela ordem_servico

A **Figura 3.26** apresenta o trecho de código que cria uma **ordem de serviço** por meio de inserção na tabela **ordem_servico**, informando as chaves estrangeiras de fazenda, centro de custo e operação, além do *status* e da identificação do talhão.

As datas são registradas no padrão ISO 8601 (ISO, 2019) e a data de fechamento é opcional. A aplicação solicita apenas o identificador gerado (**cd_ordem_servico**) como retorno, reduzindo o tráfego de dados.

```dart
// Cria uma nova Ordem de Serviço e retorna o ID (cd_ordem_servico) gerado no banco
Future<int> inserir({
  //// Parâmetros obrigatórios (FKs e dados de negócio)
  required int cdFazenda,
  required int cdCusto,
  required int cdOperacao,
  required String status,
  required String talhaoNome,
  // Campo opcional: data de fechamento (pode ser nulo)
  DateTime? dataFechamento,
}) async {
  // Insert na tabela 'ordem_servico' com os valores informados
  final tuplaRegistro = await _clienteSupabase
      .from('ordem_servico')
      .insert({
        'cd_fazenda': cdFazenda,
        'cd_custo': cdCusto,
        'cd_operacao': cdOperacao,
        'status': status,
        'talhao': talhaoNome,
        'data_cadastro': DateTime.now().toIso8601String(),
        // Se dataFechamento for nula, envia null (campo ficará vazio)
        'data_fechamento': dataFechamento?.toIso8601String(),
      })
      .select('cd_ordem_servico')
      // Garante retorno de exatamente 1 linha
      .single();
  // Converte o identificador retornado para int e devolve ao chamador
  return (tuplaRegistro['cd_ordem_servico'] as num).toInt();
}
```

Figura 3.26. Código de inserção de ordem de serviço na tabela ordem_servico. FONTE: elaborada pelo autor.

A **Figura 3.27** apresenta o trecho de código responsável pela atualização de ordens de serviço por meio do comando **update** na tabela **ordem_servico**, filtrada pela chave primária (**cd_ordem_servico**). O método recebe as chaves estrangeiras de fazenda, centro de custo e operação, o *status*, a identificação do talhão e a data de fechamento. Quando informada, a data é serializada no padrão ISO 8601; se nula, o campo permanece vazio no banco. Essa atualização preserva os vínculos de integridade entre as tabelas.

```dart
// Atualiza uma Ordem de Serviço existente (filtrando pela PK cd_ordem_servico)
Future<void> atualizar({
  // Identificador da OS a ser atualizada (chave primária)
  required int cdOs,
  // Novos valores (FKs e campos de negócio)
  required int cdFazenda,
  required int cdCusto,
  required int cdOperacao,
  required String status,
  required String talhaoNome,
  DateTime? dataFechamento,
}) async {
  await _clienteSupabase
      .from('ordem_servico')
      .update({
        'cd_fazenda': cdFazenda,
        'cd_custo': cdCusto,
        'cd_operacao': cdOperacao,
        'status': status,
        'talhao': talhaoNome,
        'data_fechamento': dataFechamento?.toIso8601String(),
      })
      // WHERE cd_ordem_servico = cdOs  (garante atualizar apenas a OS alvo)
      .eq('cd_ordem_servico', cdOs);
}
```

Figura 3.27. Código de atualização de ordem de serviço: *update*. FONTE: elaborada pelo autor.

A **Figura 3.28** apresenta o trecho de código responsável por consultar uma ordem de serviço específica por meio de busca filtrada pela chave primária (**cd_ordem_servico**). A aplicação seleciona apenas os campos necessários e utiliza **maybeSingle()** para obter, no máximo, uma linha, retornando *null* quando a ordem de serviço não é encontrada. Esse padrão reduz o tráfego de dados e simplifica o tratamento da ausência de registros.

```dart
  // Consulta uma Ordem de Serviço pelo identificador (PK)
  Future<Map<String, dynamic>?> consultarPorId(int cdOs) async {
    return await _clienteSupabase
        .from('ordem_servico')
        .select(
          'cd_ordem_servico, data_cadastro, data_fechamento, cd_operacao, cd_custo, cd_fazenda, status, talhao',
        )
        // Filtro pela chave primária (garante unicidade do resultado)
        .eq('cd_ordem_servico', cdOs)
        .maybeSingle();
  }
}
```

Figura 3.28. Código de consulta de ordem de serviço pelo identificador. FONTE: elaborada pelo autor.

3.4.5 Código de rastreamento por GPS e visualização da trilha (*OpenStreetMap*)

Para implementar a tela de rastreamento por GPS, utilizou-se o pacote *flutter_map*, uma biblioteca *open source* para o *framework Flutter* que integra mapas do *OpenStreetMap* (OSM). Essa abordagem agrega ao aplicativo funcionalidades cartográficas com dados livres e, em geral, sem necessidade de chave de *application programming interface* (API[3]) (exigindo apenas atribuição e **user-agent** conforme a

[3] **API:** conjunto de comandos e padrões que permite a comunicação entre sistemas e aplicativos.

política do OSM).

Segundo a documentação oficial do flutter_map (2024), o pacote é versátil, fácil de aprender e altamente configurável, sendo uma solução ideal para integração de mapas em aplicações *Flutter*, com suporte a diferentes fontes de *tiles*, camadas e *plugins* adicionais que expandem suas capacidades.

Segundo o OpenStreetMap (2024), a plataforma oferece dados cartográficos abertos, mantidos colaborativamente por voluntários que atualizam vias, pontos de interesse e outros elementos com apoio de imagens aéreas, GPS e verificação em campo. O conteúdo pode ser reutilizado mediante atribuição e respeito à licença de uso. A Fundação *OpenStreetMap* (OSMF) coordena as políticas e a infraestrutura do projeto.

A **Figura 3.29** apresenta o trecho de código responsável por verificar se o serviço de localização está ativado antes do início da coleta. No método **_verificarPermissoesEIniciar()**, primeiramente é feita a verificação do estado do serviço de localização do dispositivo por meio da função **serviceEnabled ()** (que retorna um *Future<bool>* do pacote *location*). Caso o serviço esteja desativado, o sistema solicita sua ativação por meio do método **requestService ()**. Se o usuário optar por não o habilitar, o fluxo é interrompido e uma mensagem informativa é exibida, orientando o operador sobre a necessidade de ativar a localização para prosseguir com a coleta.

```
48  Future<void> _verificarPermissoesEIniciar() async {
49    // Verifica se o serviço de localização do aparelho está ligado: serviceEnabled().
50    bool servicoHabilitado = await _localizacao.serviceEnabled();
51    if (!servicoHabilitado) {
52      // Se não estiver, pede para ligar: requestService()
53      servicoHabilitado = await _localizacao.requestService();
54      if (!servicoHabilitado) {
55        if (!mounted) return;
56        ScaffoldMessenger.of(context).showSnackBar(
57          const SnackBar(content: Text('Ative os serviços de localização.')),
58        );
59        return;
60      }
61    }
```

Figura 3.29. Código para verificar se o serviço de localização está ativado. FONTE: elaborada pelo autor.

A **Figura 3.30** apresenta um trecho do código onde ocorre a checagem de permissões. O *status* atual é obtido com **hasPermission()**. Se for *denied*, solicita-se

a permissão com **requestPermission()**; se o resultado não for *granted*[4], a execução é interrompida, pois não é possível prosseguir sem permissão. Se o *status* for *deniedForever* (o usuário marcou "não perguntar novamente" /bloqueou nas configurações), exibe-se a mensagem "Permissão negada permanentemente" e o processo é encerrado.

```
63    /* Esse bloco faz a verificação e o pedido de permissão de localização antes de
64    iniciar o GPS. Checa o status atual com hasPermission() */
65    PermissionStatus permissaoConcedida = await _localizacao.hasPermission();
66    // Se estiver denied (negada, mas ainda solicitável), pede a permissão com requestPermission().
67    if (permissaoConcedida == PermissionStatus.denied) {
68      permissaoConcedida = await _localizacao.requestPermission();
69      //Se o resultado não for granted, interrompe: não dá para prosseguir sem permissão.
70      if (permissaoConcedida != PermissionStatus.granted) return;
71    }
72    // Se estiver deniedForever (usuário marcou "não perguntar novamente"/bloqueou nas configurações):
73    if (permissaoConcedida == PermissionStatus.deniedForever) {
74      if (!mounted) return;
75      ScaffoldMessenger.of(context).showSnackBar(
76        const SnackBar(content: Text('Permissão negada permanentemente.')),
77      );
78      return;
79    }
```

Figura 3.30. Código para verificar se o serviço de permissão está concedido. FONTE: elaborada pelo autor.

Com o serviço ativo e a permissão concedida, configuram-se os parâmetros de coleta com *changeSettings(accuracy: LocationAccuracy.navigation, distanceFilter: 100)*, definindo alta precisão e registro de novos pontos apenas após 100 m de deslocamento. Por fim, com as condições atendidas, o método chama **_iniciarRastreamento()** (ver **Figura 3.31**).

```
81    // Esse bloco configura como o GPS vai coletar os pontos (Precisão/intervalo)
82    await _localizacao.changeSettings(
83      accuracy: LocationAccuracy.navigation,
84      distanceFilter: 100, // salva a cada ~100m
85    );
86
87    _iniciarRastreamento();
88  }
```

Figura 3.31. Código de configuração de precisão e intervalo coleta de pontos GPS. FONTE: elaborada pelo autor.

A **Figura 3.32** apresenta o trecho de código responsável pela leitura da posição inicial e início efetivo do rastreamento por GPS. No método **_iniciarRastreamento()**, realiza-se a captura imediata da posição atual do dispositivo utilizando a função **getLocation()** do pacote location, que retorna um objeto contendo latitude e longitude.

[4] **granted:** termo em inglês que significa "concedido". No Flutter, indica que a permissão solicitada pelo aplicativo foi autorizada pelo usuário.

Em seguida, cria-se a variável pontoInicial com essas coordenadas, representada pelo tipo **LatLng** da biblioteca **latlong2**, utilizada pelo componente **flutter_map** para manipulação de coordenadas geográficas.

Com o ponto inicial definido, o estado da tela é atualizado por meio de **setState()**, armazenando a posição atual **(_posicaoAtual)** e adicionando o primeiro elemento à **lista _rota**, que representa a trilha visualizada no mapa. Logo após, o ponto inicial é inserido no banco de dados *Supabase* pela chamada do método **_salvarPonto(pontoInicial)**, registrando latitude, longitude e o horário exato da coleta (**timestamp ISO 8601**).

Por fim, a variável de controle **_coletando** é definida como verdadeira, indicando que o processo de rastreamento está ativo, e o método prossegue configurando o **listener** contínuo de localização, responsável por registrar os pontos subsequentes durante o deslocamento. Caso ocorra alguma falha na inicialização (por exemplo, ausência de sinal GPS ou erro interno do *plugin*), a exceção é capturada pelo bloco **try/catch**, exibindo uma notificação ao usuário por meio de um **SnackBar** sem interromper o funcionamento do aplicativo.

```dart
Future<void> _iniciarRastreamento() async {
  try {
    final posicaoLida = await _localizacao.getLocation();
    final pontoInicial = l2.LatLng(posicaoLida.latitude!, posicaoLida.longitude!);

    //_posicaoAtual recebe pontoInicial (o marcador já aparece no mapa).
    //_rota.add(pontoInicial) inicia a polilinha (a trilha não começa "vazia").
    //Isso garante que o início do trajeto fica registrado
    setState(() {
      _posicaoAtual = pontoInicial;
      _rota.add(pontoInicial);
    });

    // Salva o primeiro ponto no backend
    await _salvarPonto(pontoInicial);
    // Após o primeiro ponto, define _coletando = true e assina onLocationChanged.
    /* Daqui em diante, cada nova amostra repete o ciclo: valida → adiciona
    em _rota → chama _salvarPonto(...) → move a câmera mantendo o zoom atual.*/
    _coletando = true;
    _localizacao.onLocationChanged.listen((LocationData data) async {
      if (data.latitude == null || data.longitude == null) return;

      final ponto = l2.LatLng(data.latitude!, data.longitude!);
      setState(() {
        _posicaoAtual = ponto;
        _rota.add(ponto);
      });

      await _salvarPonto(ponto);

      // move o mapa mantendo o zoom atual
      final zoomAtual = _controladorMapa.camera.zoom;
      _controladorMapa.move(ponto, zoomAtual);
    });
  } catch (e) {
    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Erro ao iniciar GPS: $e')),
    );
  }
```

Figura 3.32. Código Responsável pela leitura da posição inicial e início do rastreamento por GPS.
FONTE: elaborada pelo autor.

A **Figura 3.33** apresenta o trecho de código responsável pela renderização dinâmica do mapa interativo e exibição dos elementos geográficos na interface do aplicativo por meio do pacote **flutter_map**. Inicialmente, o código realiza uma verificação condicional para garantir que a posição atual do dispositivo esteja disponível. Caso o valor da variável posicao seja nulo, significa que a localização ainda está sendo obtida, e, portanto, é exibido ao usuário um indicador de carregamento centralizado (**CircularProgressIndicator()**), informando que o sistema segue em processamento.

Quando a posição é carregada com sucesso, o componente *FlutterMap* é inicializado, utilizando um *MapController* para possibilitar o controle programático do mapa, além de configurar as opções iniciais de visualização, como *initialCenter* com a posição atual do usuário e *initialZoom* com valor 18, oferecendo um nível de aproximação adequado para navegação urbana. Em seguida, o *TileLayer* é adicionado para carregar os blocos de mapa (*tiles*) provenientes do provedor *OpenStreetMap*, utilizando uma URL[5] pública do serviço e um cabeçalho HTTP[6] personalizado (*User-Agent*) para identificação da aplicação conforme as diretrizes de uso da API do OSM. O parâmetro *maxZoom* limitado a 19 impede a tentativa de solicitações além do nível máximo permitido pelo provedor de mapas.

Complementando a camada base do mapa, o código inclui ainda um *PolylineLayer*, responsável pela renderização da trilha percorrida pelo usuário, representada pela lista de pontos armazenada na **variável _rota**.

[5] **URL (*Uniform Resource Locator*)** é o endereço utilizado para localizar e acessar recursos na internet, como páginas, arquivos, imagens ou serviços. É o identificador que indica onde o recurso está hospedado e como ele deve ser acessado.

[6] **HTTP (*Hypertext Transfer Protocol*)** é o protocolo responsável pela comunicação entre clientes e servidores na web. Ele define como as requisições são enviadas e como as respostas são recebidas, permitindo o carregamento de páginas, imagens, textos e outros recursos na internet.

```
177       body: posicao == null
178           ? const Center(child: CircularProgressIndicator())
179           // Quando a posição está disponível, inicializa o mapa interativo.
180           : FlutterMap(
181               mapController: _controladorMapa,
182               options: MapOptions(
183                 initialCenter: posicao, // Define a posição inicial do mapa.
184                 initialZoom: 18, // Nível de zoom adequado para navegação
185               ), // MapOptions
186               children: [
187                 TileLayer(
188                   // Camada de blocos de mapa (tiles) provenientes do OpenStreetMap.
189                   urlTemplate:
190                       'https://tile.openstreetmap.org/{z}/{x}/{y}.png',
191                   tileProvider: NetworkTileProvider(
192                     // Identificação da aplicação conforme política do OSM.
193                     headers: {'User-Agent': userAgent},
194                   ), // NetworkTileProvider
195                   maxZoom: 19, // Limite máximo de aproximação permitido pelo provedor.
196                 ), // TileLayer
197                 // Camada responsável por desenhar a trilha percorrida pelo usuário.
198                 PolylineLayer(
199                   polylines: [
200                     Polyline(
201                       points: _rota,
202                       strokeWidth: 5,
203                       color: Colors.blue,
204                     ), // Polyline
205                   ],
206                 ), // PolylineLayer
```

Figura 3.33. Trecho de código responsável pela exibição do mapa com base no *OpenStreetMap* utilizando pacote *FlutterMap*. FONTE: elaborada pelo autor.

3.5 Interface do protótipo

Esta seção apresenta as algumas telas do protótipo desenvolvido, com o objetivo de demonstrar a interface gráfica e a interação do usuário com o sistema. As imagens a seguir ilustram o fluxo de navegação, desde a autenticação até o acompanhamento das funcionalidades de rastreamento e visualização de dados no mapa.

As telas foram estruturadas para garantir uma navegação intuitiva, com botões de ação claros e elementos visuais padronizados conforme a identidade do sistema.

A **Figura 3.34** apresenta a interface de *login* sob a perspectiva do usuário ao acessar o sistema por meio do ambiente *desktop*. Nessa tela, o sistema solicita o preenchimento dos campos usuário e senha para autenticação. Caso o usuário acione o botão “Acesso” sem preencher os campos obrigatórios, o sistema exibirá a mensagem “Informe usuário e senha”. Caso algum dos campos seja preenchido de forma incorreta, uma mensagem de erro será apresentada. Se o nome de usuário estiver correto, porém a senha incorreta, o sistema exibirá a mensagem “Senha

incorreta”, indicando falha na autenticação.

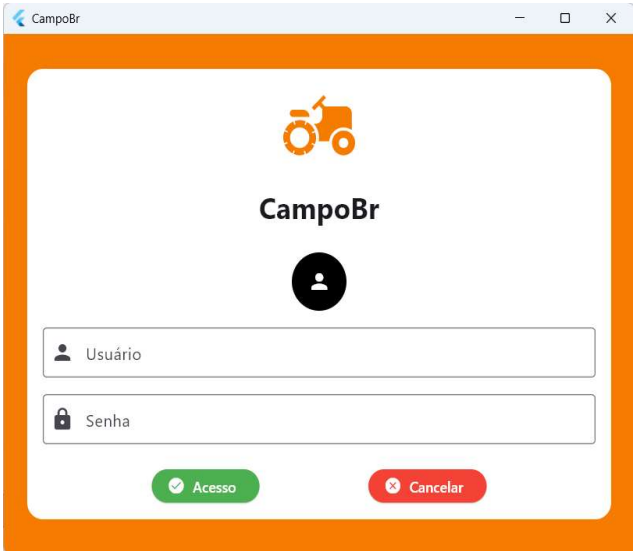

Figura 3.34. Interface de autenticação de usuário. FONTE: elaborada pelo autor.

Após a autenticação bem-sucedida, o usuário é direcionado para a tela principal do sistema (ver **Figura 3.35**).

O menu dessa interface é composto por botões que permitem o acesso direto aos principais módulos do sistema, como cadastro de tipo de usuário, cadastro de *login* e senha, cadastro de centro de custo, cadastro fazenda, cadastro de talhão, cadastro operador, cadastro de equipamento e cadastro de operação.

Além disso, o sistema disponibiliza opções para abertura de ordem de serviço, permitindo que o administrador programe as atividades que serão executadas na fazenda, incluindo o local onde cada tarefa será realizada. Também são oferecidas funções de abertura de apontamento e acesso a relatórios, que possibilitam o gerenciamento e o acompanhamento das atividades agrícolas registradas. Essa tela funciona como o ponto central de navegação do sistema, reunindo os principais recursos de operação e controle.

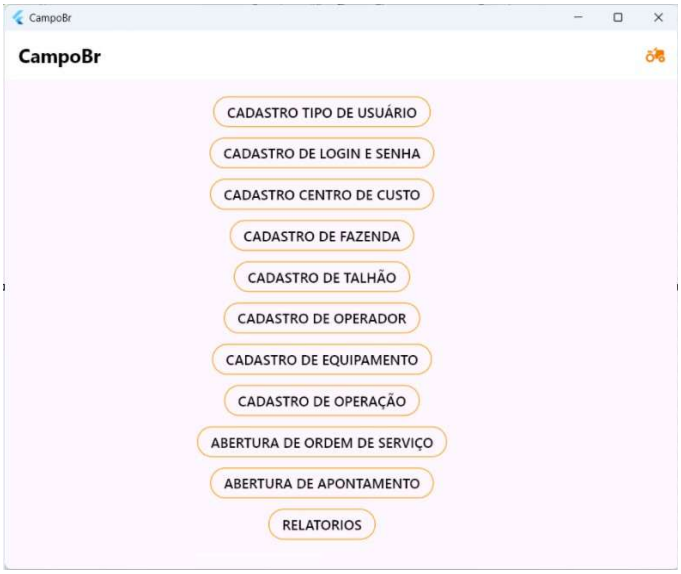

Figura 3.35. Interface do menu principal do sistema. FONTE: elaborada pelo autor.

A **Figura 3.36** apresenta a interface destinada ao cadastro de operadores no sistema. Nessa tela, o usuário pode inserir um novo registro, alterar informações de um operador já cadastrado ou realizar consultas utilizando o número de matrícula do operador como critério de busca.

Quando o usuário opta por cadastrar um novo operador, os campos número de matrícula e data do cadastro são gerados automaticamente pelo sistema, enquanto os campos nome e cargo são de preenchimento obrigatório pelo usuário. Caso seja realizada uma consulta com um número de matrícula inexistente, o sistema exibirá uma mensagem informativa na tela, alertando sobre a ausência de registro correspondente.

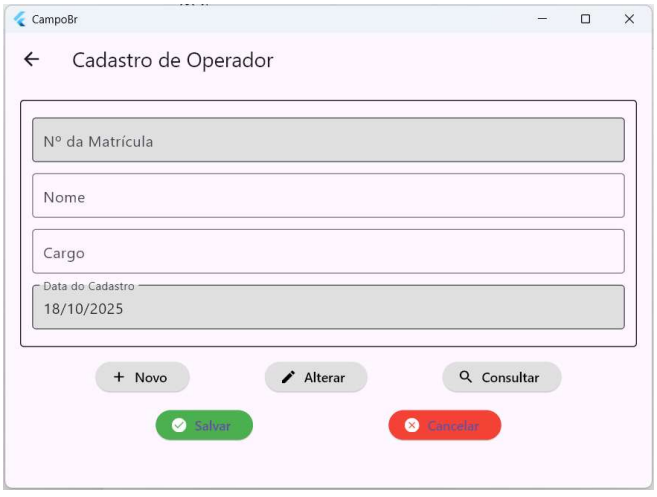

Figura 3.36. Interface cadastro de operador. FONTE: elaborada pelo autor.

A **Figura 3.37** apresenta a interface destinada à abertura de ordens de serviço no sistema. Essa tela permite ao usuário registrar novas ordens de serviço, realizar alterações ou consultar registros existentes

Para o cadastro de uma nova ordem, os campos que requerem preenchimento manual pelo usuário são: **código da fazenda**, **código do centro de custo**, **código da operação** e **talhão**. O campo da data de fechamento é opcional, podendo ser deixado em branco quando ainda não houver previsão de término da atividade.

Os demais campos são preenchidos automaticamente pelo sistema, incluindo informações como número da ordem de serviço, *status*, data de abertura, fazenda, centro de custo, operação e área do talhão. Essa automatização tem como objetivo reduzir erros de digitação, agilizar o processo de registro e garantir maior integridade dos dados inseridos.

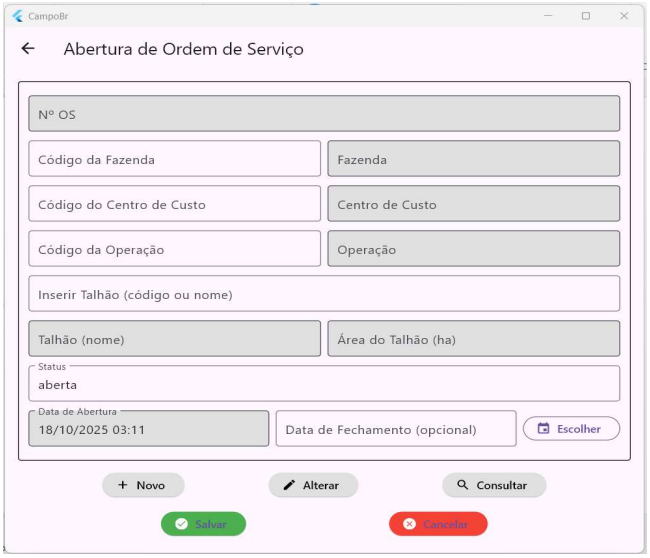

Figura 3.37. Interface abertura de ordem de serviço. FONTE: elaborada pelo autor

A **Figura 3.38** apresenta a interface destinada ao usuário operador, visualizada na versão Android do sistema. Essa tela corresponde ao início do processo de abertura de apontamento, no qual o operador deve informar seu número de matrícula e inserir o número do equipamento que está utilizando na atividade do dia.

Após preencher os campos obrigatórios, o operador deve acionar o botão “Próximo”, que dará continuidade ao processo de registro, exibindo o cabeçalho do apontamento, composto pelos campos: matrícula do operador, nome do operador, número do equipamento e descrição do equipamento.

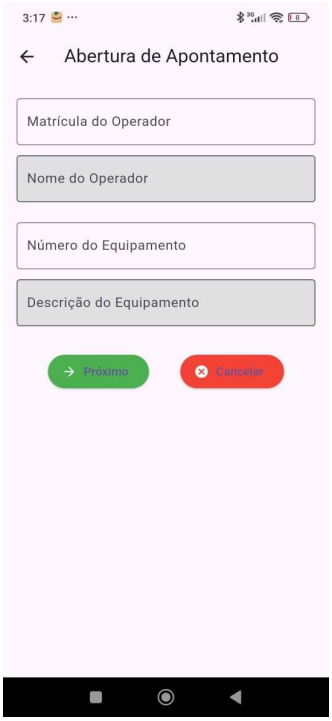

Figura 3.38. Interface abertura de apontamento. FONTE: elaborada pelo autor.

A **Figura 3.39** apresenta a tela do boletim de apontamento em andamento, na qual são exibidas as informações do cabeçalho do registro. Entre os dados apresentados, destacam-se o número do boletim, o nome do operador, a descrição da frota e a data de abertura do apontamento.

Ao acionar o botão “Adicionar Linha”, novos campos são exibidos na tela para que o operador insira as informações correspondentes à atividade realizada. O usuário deve preencher o campo “Nº OS” (número da ordem de serviço), enquanto os demais campos: Centro de Custo (CC), Fazenda, Número da Operação e Descrição da Operação são preenchidos automaticamente pelo sistema.

Antes de iniciar a execução da atividade, o operador deve registrar o valor do horímetro inicial, obtido diretamente no painel do equipamento em uso. O campo horímetro final deve permanecer em branco, sendo preenchido apenas ao término da operação.

Com o número da ordem de serviço inserido, o usuário deve acionar o botão “GPS” para que o sistema realize a verificação da localização e a coleta dos pontos da área trabalhada. Após a conclusão da atividade, o operador deve selecionar a opção “Finalizar” para encerrar o apontamento e salvar os dados registrados.

Figura 3.39. Interface de adição de OS no boletim de apontamento. FONTE: elaborada pelo autor.

A **Figura 3.40** apresenta a interface de rastreamento GPS em tempo real do sistema. Essa funcionalidade permite que o operador acompanhe o deslocamento do equipamento agrícola durante a execução da atividade no campo, exibindo sua posição atual sobre o mapa.

Por meio dessa tela, o sistema realiza a coleta contínua dos pontos geográficos (coordenadas GPS), possibilitando o monitoramento preciso da rota percorrida pelo equipamento. As informações obtidas são armazenadas no banco de dados do sistema, permitindo a geração de relatórios para análise posterior das operações realizadas.

Além disso, essa interface foi projetada para oferecer uma visualização clara e intuitiva da movimentação do equipamento, utilizando marcadores e elementos gráficos que facilitam a identificação do trajeto e da direção deslocada.

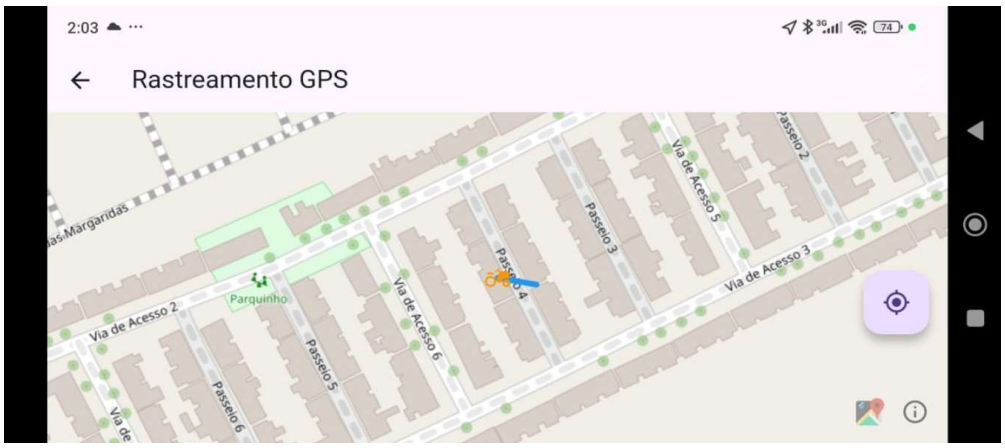

Figura 3.40. Interface de rastreamento GPS em tempo real. FONTE: elaborada pelo autor.

3.6 Teste

Durante o desenvolvimento foram feitos testes de funcionamento para verificar se cada parte do sistema estava operando corretamente e se os dados eram gravados no banco *Supabase*.

Os testes foram feitos em ambiente de simulação, sem uso em fazenda real. Os principais resultados foram:

● *Login* e autenticação: o sistema validou corretamente usuário e senha criptografada;

● Cadastros: foi possível inserir, alterar e consultar dados de fazendas, talhões, operadores, equipamentos, centros de custo e operações, com gravação correta no banco;

● Ordens de serviço: a abertura de novas ordens funcionou normalmente, com validação dos campos e preenchimento automático de informações;

● Apontamento: o aplicativo *Android* registrou as informações do operador, equipamento e horímetros inicial e final sem falhas;

● Rastreamento GPS: o sistema coletou corretamente os pontos de localização e exibiu em tempo real o traçado do deslocamento do equipamento, representando o percurso realizado durante a operação. Os pontos coletados foram salvos no banco de dados, permitindo o uso posterior dessas informações para consultas.

Mesmo sendo um protótipo, o sistema apresentou um bom desempenho e cumpriu as funções planejadas. O uso do *Supabase*, que utiliza o banco *PostgreSQL* em nuvem, garantiu sincronização estável entre as versões *Android* e *Windows*, com baixo custo de operação.

4. Conclusão

O desenvolvimento do protótipo teve como objetivo principal criar um *software* multiplataforma para apoio das atividades agrárias, integrando as informações do campo e do escritório de forma prática e acessível. O projeto atendeu ao propósito de disponibilizar uma ferramenta de baixo custo, funcional e fácil de usar, voltada especialmente a pequenos e médios produtores rurais.

Durante o desenvolvimento foram aplicados conceitos de geolocalização (GPS), banco de dados em nuvem (*Supabase*) e programação multiplataforma (*Flutter*). O sistema foi estruturado para permitir o cadastro e gerenciamento de informações agrícolas, como fazendas, talhões, operadores, equipamentos, centros de custo, ordens de serviço e apontamentos.

A integração entre o aplicativo *Android* e o sistema para *desktop Windows* resultou em uma solução completa, onde o campo e o escritório se comunicam de forma sincronizada.

Os testes realizados em ambiente de simulação mostraram que o protótipo funciona de forma estável, segura e com armazenamento correto dos dados. As funcionalidades de *login*, cadastro, apontamento e rastreamento GPS atenderam às expectativas, confirmando a viabilidade do projeto e sua possível aplicação em ambiente real.

Mesmo com os bons resultados, foram identificadas possibilidades de melhoria, como a criação de um modo *offline* no aplicativo *Android*, o uso do *Supabase Auth* para autenticação, o aprimoramento das consultas e o aperfeiçoamento do módulo de apontamento, que poderá incluir cálculos automáticos de distância percorrida e estimativa da área trabalhada. Essas melhorias podem deixar o sistema mais eficiente, prático e completo em versões futuras

Conclui-se que o protótipo é uma ferramenta promissora para a gestão e monitoramento de atividades agrícolas, contribuindo para a modernização das práticas no setor e favorecendo a inclusão tecnológica no meio rural. O protótipo apresenta potencial para evoluir e se tornar uma solução completa de apoio à tomada de decisão e controle operacional nas propriedades rurais.

4.1 Trabalhos futuros

Durante o desenvolvimento do protótipo, foram identificadas algumas melhorias que podem ser implementadas em trabalhos futuros, permitindo planejar ampliações que contribuam ainda mais para a evolução e o aprimoramento do sistema.

Uma das principais propostas é o desenvolvimento do modo *offline*, permitindo que o aplicativo *Android* funcione mesmo sem acesso à *internet*. Nessa versão, os dados seriam armazenados localmente e enviados ao *Supabase* assim que uma conexão estável fosse restabelecida. Essa melhoria tornará o sistema mais adequado à realidade do campo, onde a instabilidade de rede é comum.

Também está prevista a migração do processo de autenticação para o sistema de *login* nativo do *Supabase Auth*, garantindo maior segurança e simplificando o controle de acessos. Além disso, a criação de novas tabelas de funcionários e cargos poderá ampliar o gerenciamento do quadro de pessoal da fazenda, tornando o sistema mais completo e integrado.

Além disso, recomenda-se a criação de duas novas tabelas — funcionários e cargos — acompanhada da remoção da tabela operador do banco de dados. A tabela funcionário permitirá o registro completo das informações de todos os colaboradores da fazenda, enquanto a tabela cargos armazenará as funções existentes e manteria vínculo direto com cada funcionário. Com essa estrutura, será possível identificar com precisão o cargo ocupado por cada colaborador, como assistente administrativo, operador de máquinas, supervisor ou gerente. Essa melhoria ampliará o controle sobre o quadro de pessoal e tornaria o sistema mais completo e eficiente para a gestão da fazenda.

Outra proposta é o aperfeiçoamento do módulo de apontamento, que poderá incluir novas funções, como o cálculo automático da distância percorrida pelo equipamento e a estimativa da área total trabalhada. Essas funcionalidades possibilitarão um controle mais preciso do desempenho operacional e da produtividade em campo.

Também propõe o desenvolvimento de relatórios automáticos e gráficos de desempenho, que permitam ao gestor visualizar indicadores de produtividade, utilização de equipamentos e desempenho dos operadores.

Por fim, uma evolução importante seria a integração com sensores agrícolas e

tecnologias de *Internet* das Coisas (IoT), permitindo que o sistema receba dados diretamente das máquinas ou sensores de campo. Essa integração possibilitará análises mais precisas e a tomada de decisões mais rápidas e assertivas nas atividades agrícolas.

Referências bibliográficas

ABRAHAM, Ajith; JAIN, Lakhmi C**. Inteligência Artificial na Agricultura**. Springer, 2021.

ALVES, Daniele Barroca Marra. **GNSS: Conceitos, Modelagem e Perspectivas Futuras do Posicionamento por Satélite**. UNESP, 2013. Disponível em: https://www.fct.unesp.br/Home/Pesquisa/GEGE/daniele_barroca_reuniao_gege_12042013.pdf. Acesso em: 11 mar. 2025.

BOCHTIS, Dionysis. **Automação na Agricultura: Garantindo o Abastecimento de Alimentos para as Futuras Gerações.** Springer, 2020.

CORREA, Marques Priscila. **Topografia e geoprocessamento**. 1. ed. Porto Alegre: SAGAH, 2017.

DART. Dart – **Uma linguagem de programação portável e produtiva para aplicações de alta qualidade**. [S.l.]: Dart Developers, [2025]. Disponível em: https://dart.dev/. Acesso em: 20 out. 2025.

ELMASRI, Ramez.; NAVATHE, Shamkant. B. **Sistemas de Banco de Dados**. 6. ed. São Paulo: Pearson, 2011.

FLUTTER_MAP. flutter_map – **A versatile mapping package for Flutter**. 2024. Disponível em: https://pub.dev/packages/flutter_map. Acesso em: 14 jan. 2025.

GARCÍA ÁLVAREZ, David. **Sistema GNSS (Sistema Global de Navegação por Satélite).** Trabalho de Conclusão de Curso (Graduação em Engenharia). Universidade de Jaén, 2009.

GOOGLE. **Build for multiple platforms from codebase with Flutter**. [S.l.]: Google, [2025]. Disponível em: https://flutter.dev/multi-platform. Acesso em: 20 out. 2025.

HUANG, Yanbo. **Robôs Agrícolas: Mecanismos e Prática.** CRC Press, 2021.

NASCIMENTO, José Alexandre. **Geoprocessamento Aplicado à Agricultura.** Editora UFV,2017.

OPENSTREETMAP. **Sobre OpenStreetMap**. 2024. Disponível em: https://www.openstreetmap.org/about. Acesso em: 14 jan. 2025.

RR TECNOLOGIA AGRÍCOLA. **Conheça nossa automação agrícola e eleve sua produtividade na próxima colheita**. Dourados-MS: RR Tecnologia Agrícola, 2024. Disponível em: https://rrtecnologiaagricola.com.br/. Acesso em: 20 out. 2025.

TOTVS. Sistema PIMS: **A tecnologia revoluciona as empresas**. São Paulo: TOTVS, 2021. Disponível em: https://www.totvs.com/blog/gestao-agricola/sistema-pims/. Acesso em: 20 out. 2025.

ISO. ISO 8601:2019 – Date and time — **Representations for information interchange**. Geneva: International Organization for Standardization, 2019.