INSTITUTO FEDERAL
Rondônia

MINISTÉRIO DA EDUCAÇÃO
Secretaria de Educação Profissional e Tecnológica
Instituto Federal de Educação, Ciência e Tecnologia de Rondônia

Campus Vilhena

Coordenação do Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas

EDUARDO DOS SANTOS TARTAS

Sistema de Apoio ao Manejo de Gado de Corte: Controle de Lotes e Pastagens em Pequenas Propriedades

VILHENA – RO

2026

EDUARDO DOS SANTOS TARTAS

Sistema de Apoio ao Manejo de Gado de Corte: Controle de Lotes e Pastagens em Pequenas Propriedades

Artigo entregue como Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO), Campus Vilhena, como requisito parcial para obtenção do grau de Tecnólogo, junto ao Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do professor Mestre Gilberto Pereira da Silva.

VILHENA – RO

2026

Ficha catalográfica elaborada pelo Sistema Gerador de Ficha Catalográfica do IFRO.

Tartas, Eduardo dos Santos.
Sistema de apoio ao manejo de gado de corte: controle de lotes e pastagens em pequenas propriedades / Eduardo dos Santos Tartas. - Vilhena, 2026.
29 f. : il.

Orientador(a): Prof. Me. Gilberto Pereira da Silva.

Trabalho de Conclusão de Curso (Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO, Vilhena, 2026.

1. Bovinocultura de corte. 2. Aplicativo móvel. 3. Pastagem rotacionada. 4. Offline-first. 5. Sincronização de dados. I. Silva, Gilberto Pereira da (orient.). II. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO. III. Título.

Bibliotecário(a) Responsável: Rosilene Maria do Couto Marques, CRB-11/321

EDUARDO DOS SANTOS TARTAS

Sistema de Apoio ao Manejo de Gado de Corte: Controle de Lotes e Pastagens em Pequenas Propriedades

Artigo entregue como Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO), Campus Vilhena, como requisito parcial para obtenção do grau de Tecnólogo, junto ao Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do professor Mestre Gilberto Pereira da Silva.

Aprovado em 05/11/2026 pela banca examinadora.

gov.br
Documento assinado digitalmente
MARCO ANTONIO AUGUSTO DE ANDRADE
Data: 06/11/2026 15:00:34-0300
Verifique em https://validar.iti.gov.br

Prof. Mestre Marco Antônio Augusto de Andrade
Examinador interno

gov.br
Documento assinado digitalmente
WESLEY JHONNES RAMOS ROLIM
Data: 06/11/2026 15:03:48-0300
Verifique em https://validar.iti.gov.br

Prof. Mestre Wesley Jhonnes Ramos Rolim
Examinador interno

gov.br
Documento assinado digitalmente
GILBERTO PEREIRA DA SILVA
Data: 06/11/2026 15:07:38-0300
Verifique em https://validar.iti.gov.br

Prof. Mestre Gilberto Pereira da Silva
Orientador

Resumo

A gestão de pequenas propriedades de bovinocultura de corte enfrenta a falta de registro sistemático e a conectividade restrita do meio rural, o que dificulta o acompanhamento das pastagens rotacionadas. Este trabalho tem como objetivo desenvolver o Pasto Livre, um aplicativo móvel offline-first para Android que integra o controle de rebanho por lotes, o histórico de manejo das pastagens e o inventário de insumos. A pesquisa é aplicada e seguiu a Design Science Research, com requisitos levantados junto a um produtor rural. O aplicativo, em Flutter com SQLite, grava os dados no aparelho e os envia, por uma fila de saída (outbox), a uma API REST em Node.js e PostgreSQL quando há conexão. A avaliação foi técnica, sem usuários finais, com 663 testes automatizados do aplicativo e 1.050 da API e quatro cenários de desconexão em um aparelho físico. Os 17 requisitos funcionais foram atendidos, as mutações criadas sem conexão chegaram ao servidor sem perda e na ordem declarada, e uma mutação recusada não deixou registro órfão. Fora dos critérios, a edição concorrente do mesmo registro foi detectada pelo servidor, sem sobrescrever a edição mais recente. Os resultados indicam que a arquitetura é tecnicamente viável para a gestão pecuária de pequeno porte. Para o produtor, o aplicativo reúne a visão da propriedade e os avisos de descanso, ocupação e lotação, efeito que não foi medido, e a validação com produtores fica como trabalho futuro.

Palavras-chave: *bovinocultura de corte; aplicativo móvel; pastagem rotacionada; offline-first; sincronização de dados.*

Abstract

The management of small beef cattle farms faces a lack of systematic record keeping and limited rural connectivity, which makes it hard to monitor rotational pastures. This paper aims to develop Pasto Livre, an offline-first mobile application for Android that integrates herd control by lots, pasture management history, and input inventory. The research is applied and followed Design Science Research, with requirements gathered from a rural producer. The application, built with Flutter and SQLite, saves data on the device and sends it, through an outbox queue, to a REST API in Node.js and PostgreSQL when a connection is available. The evaluation was technical, without end users, with 663 automated tests of the application and 1,050 of the API and four disconnection scenarios on a physical device. The 17 functional requirements were met, the mutations created offline reached the server without loss and in the declared order, and a rejected mutation left no orphan record. Outside the criteria, concurrent editing of the same record was detected by the server, without overwriting the newer edit. The results indicate that the architecture is technically viable for small-scale livestock management. For the producer, the application brings together an overview of the farm and alerts on rest, occupation, and stocking rate, an effect that was not measured, and validation with producers is left as future work.

Keywords: *beef cattle farming; mobile application; rotational grazing; offline-first; data synchronization.*

1. Introdução

A bovinocultura de corte ocupa posição central na economia brasileira, e o país é o maior produtor mundial de carne bovina (ABIEC, 2026). Segundo o Centro de Estudos Avançados em Economia Aplicada (CEPEA, 2026), o agronegócio respondeu por 25,13% do Produto Interno Bruto nacional em 2025. Esse peso econômico, porém, não se traduz em gestão profissionalizada em todas as escalas. Nas pequenas propriedades, a maioria dos produtores não mantém controle nem planejamento formal, e o pouco que registra fica em apontamentos avulsos ou na memória (Lopes; Costa, 2024). Sem um método adequado para registrar os custos, o produtor perde o controle sobre os próprios ganhos (Lopes; Costa, 2024).

A falta de registro pesa de forma concreta no manejo das pastagens. O pastejo rotacionado alterna ocupação e repouso entre os piquetes, e o repouso é o que permite à forrageira se recuperar (Mendes; Martins; Freire, 2022, p. 2). Respeitar esse repouso pressupõe saber quando cada piquete foi ocupado e desocupado, o que é uma premissa deste trabalho. Sem esse controle, o descanso pode não ser cumprido e a lotação sair do ajuste, e a pressão excessiva dos animais sobre a área compromete o solo e as plantas forrageiras (Freitas *et al.*, 2024, p. 4). A forragem disponível influencia diretamente o ganho de peso dos animais (Mendes; Martins; Freire, 2022, p. 3). Falta, com frequência, assistência técnica para orientar esse manejo (Freitas *et al.*, 2024, p. 4).

A digitalização do agronegócio, associada à agricultura 4.0, avança de forma desigual e alcança pouco o produtor de pequeno porte (Buainain; Cavalcante; Consoline, 2021, p. 7). Sistemas comerciais consolidados, como iRancho[1], JetBov[2], são de difícil uso para o produtor com pouca escolaridade ou pouca experiência com tecnologia (Costa *et al.*, 2024). O Bovexo[3] é outro sistema da mesma categoria. Soma-se a isso uma restrição de infraestrutura: no Censo Agropecuário de 2017, apenas 1,43 milhão dos 5,07 milhões de estabelecimentos agropecuários brasileiros (28,2%, razão calculada pelo autor) declararam acesso à internet (IBGE, 2019a). Esse quadro dificulta o uso de sistemas dependentes de conexão constante.

Entre os trabalhos consultados sobre gestão pecuária em dispositivos móveis, o de Chacon exige a internet em algum ponto do fluxo (Chacon; Marinho; Alves, 2023). O de Silva dispensa a conexão após o login (Silva, 2022). O de Freitas opera sem rede, com foco no auxílio ao pastejo rotacionado (Freitas *et al.*, 2024). Permanece em aberto, entre esses trabalhos, uma ferramenta que una o controle de lotes ao manejo das pastagens e tenha a operação sem conexão como regra.

Diante desse cenário, a pesquisa responde a uma questão: como projetar e implementar um aplicativo móvel *offline-first* que integre lotes, pastagens rotacionadas e insumos e mantenha consistentes os dados sincronizados, diante da perda e do retorno da conexão?

Como resposta a essa questão, o presente trabalho propõe o Pasto Livre, um aplicativo móvel de gestão para pequenas propriedades rurais, construído com o *framework* Flutter e distribuído para Android. Para funcionar onde a conexão é escassa, a aplicação adota arquitetura *offline-first*: opera sem internet e

[1] IRANCHO. *iRancho: software para bovinos de corte*. Disponível em: https://www.irancho.com.br/. Acesso em: 1 out. 2026.

[2] JETBOV. *JetBov: software de gestão para fazendas de pecuária de corte*. Disponível em: https://jetbov.com/. Acesso em: 1 out. 2026.

[3] BOVEXO. *Bovexo: plataforma de gestão para pecuária de corte*. Disponível em: https://www.bovexo.com/. Acesso em: 1 out. 2026.

armazena os dados no próprio dispositivo, que os sincroniza com um serviço remoto quando a rede fica disponível.

O objetivo geral é desenvolver um aplicativo móvel *offline-first* que integre o controle de rebanho por lotes, o histórico de manejo das pastagens rotacionadas e o controle de insumos, com indicadores de apoio à decisão do pequeno pecuarista. Como objetivos específicos, o trabalho se propõe a:

- levantar os requisitos da gestão de pequenas propriedades pecuárias, em conversa com um produtor, e as restrições de conectividade do meio rural, na literatura;
- modelar uma estrutura de dados local para propriedades, pastagens, lotes e movimentações, com operação sem conexão;
- implementar o controle de rebanho por lotes, com o registro rastreável das movimentações entre pastos e dos manejos sanitários;
- implementar o registro do histórico de manejo das pastagens, incluindo ocupação e descanso dos piquetes;
- implementar o módulo de inventário de insumos com cálculo de autonomia da suplementação;
- implementar a sincronização dos dados com uma *Application Programming Interface* (API) REST quando houver conexão;
- verificar, por meio de testes funcionais e cenários controlados de desconexão, o atendimento aos requisitos e a consistência da sincronização dos dados.

O trabalho se justifica em dois planos. No plano prático, oferece ao pequeno produtor uma alternativa de gestão digital compatível com a baixa conectividade e o custo restrito do meio rural. O público-alvo é o pequeno pecuarista do Cone Sul rondoniense (microrregiões de Vilhena e de Colorado do Oeste), onde havia cerca de 2,3 milhões de bovinos em 2024 (IBGE, 2025). Em 2017, 8.303 dos 10.967 estabelecimentos agropecuários da região (75,7%, razão calculada pelo autor) eram de agricultura familiar (IBGE, 2019b), valor que apenas aproxima o perfil do público-alvo, pois abrange todos os estabelecimentos da região. No plano acadêmico, consolida as competências de engenharia de software do curso na construção de um artefato completo e funcional, do projeto à implantação.

2. Fundamentação Teórica

2.1. Gestão e Controle na Pecuária Familiar

O acompanhamento de uma propriedade familiar de 14,5 hectares ilustra como o controle do rebanho pode ser estruturado (Cruz *et al.*, 2022). A primeira medida adotada ali foi uma planilha simples que reunia, para cada animal, do fornecedor e da raça até a data de entrada e de saída da propriedade. Sem um registro estruturado como esse, informações básicas sobre a origem e a movimentação de cada animal ficam dispersas em anotações soltas ou dependem apenas da memória do produtor.

O registro do rebanho se completa com o controle financeiro. Em geral, as propriedades rurais seguem sem acompanhar a gestão nem planejar a atividade, e a razão apontada é que o produtor desconhece o que a contabilidade rural poderia acrescentar ao resultado do negócio (Lopes; Costa, 2024, p. 36). Gerenciar custos é planejá-los e dispor de instrumentos para acompanhá-los, o que dá mais controle sobre a

atividade e repercute na situação financeira da propriedade (Lopes; Costa, 2024, p. 36). Para apoiar essa gestão, o aplicativo desenvolvido acompanha o gasto com insumos e a receita da venda de animais.

Este trabalho parte da premissa de que o estoque de suplementação, como o sal mineral, diminui à medida que o rebanho o consome. Saber por quantos dias o saldo atende ao consumo diário, ou seja, a autonomia da suplementação, permite planejar a compra antes que o produto falte. Essa premissa sustenta o quinto objetivo específico e o cálculo de autonomia do módulo de insumos (Seção 4.2).

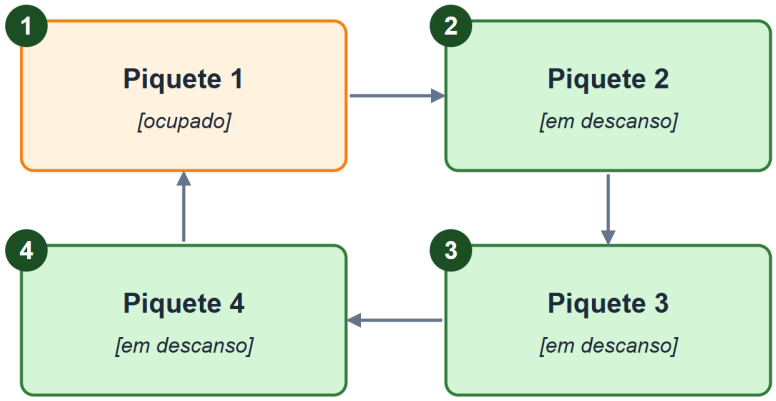
A rastreabilidade é outro registro com o qual a gestão pecuária precisa lidar. O Brasil mantém desde 2002 o Sisbov, de adesão em regra voluntária (CNA, 2024, p. 1). Em dezembro de 2024, foi lançado o Plano Nacional de Identificação Individual de Bovinos e Búfalos (PNIB), que prevê a identificação obrigatória após uma transição de oito anos, com adesão voluntária nesse período (CNA, 2024, p. 3-4). A finalidade do plano é sanitária, mas a nota técnica que o apresenta aponta um ganho de gestão: informações mais exatas para decidir sobre o manejo, a reprodução e a nutrição (CNA, 2024, p. 2). No aplicativo desenvolvido, o rebanho é organizado em lotes, cada um com a quantidade de cabeças, a movimentação entre pastos e os manejos sanitários.

Há, por fim, a lotação das pastagens, medida por hectare, que consta da lista de indicadores da Embrapa para a pecuária de corte e mede diretamente a produtividade da atividade (Costa *et al.*, 2018, p. 22-23). Este trabalho parte da premissa de que acompanhá-la com fidelidade exige o registro de quais lotes ocuparam cada pasto e por quanto tempo.

2.2. Fundamentos do Manejo de Pastagens

O pastejo rotacionado divide a área de pastagem em piquetes, e o rebanho ocupa um de cada vez enquanto os demais permanecem em descanso. Esse descanso não é tempo ocioso: nesse intervalo a forrageira se recupera do corte, o que mantém a produção do pasto por mais tempo (Mendes; Martins; Freire, 2022, p. 2). O efeito da falta desse ciclo é cumulativo. Em piquetes mal dimensionados, a pressão do rebanho se concentra em poucos pontos da área, e o consumo excessivo da forragem abre caminho para a degradação e para as plantas invasoras (Braga *et al.*, 2020).

Figura 1. Esquema do revezamento entre piquetes no pastejo rotacionado



Fonte: Elaborado pelo autor (2026). As setas indicam a ordem em que o rebanho muda de piquete.

A Figura 1 ilustra esse revezamento entre piquetes: o rebanho ocupa um deles enquanto os demais

descansam.

O dimensionamento dos piquetes segue, na teoria, um cálculo objetivo. O número de piquetes (N) depende de dois tempos. O período de descanso (PD) é o de que a forrageira precisa para se recuperar, e o período de ocupação (PO) é o que o rebanho passa em cada piquete. N é a razão entre os dois, somada ao número de lotes (L) em rodízio. Quando há um só lote, o acréscimo é de uma única unidade (Braga et al., 2020, p. 7; Martha Júnior et al., 2003, p. 4). A Equação 1 resume essa relação.

$$N = \frac{PD}{PO} + L \tag{1}$$

em que: N é o número de piquetes; PD, o período de descanso, em dias; PO, o período de ocupação, em dias; e L, o número de lotes em rodízio.

Com descanso de 35 dias e ocupação de 7 dias, por exemplo, um lote exige $35/7 + 1 = 6$ piquetes, cinco em descanso e um em ocupação. Se a ocupação cair para 5 dias, com o mesmo descanso, o resultado sobe para 8 piquetes (Braga *et al.*, 2020, p. 7). O período de descanso, por sua vez, não é fixo: varia com o clima, com o solo e com a intensidade de pastejo (Martha Júnior et al., 2003, p. 3). Ele também muda com a época do ano, sendo mais curto na fase de crescimento acelerado da forrageira e mais longo quando o crescimento desacelera (Braga *et al.*, 2020, p. 10). Este trabalho parte da premissa de que isso torna o acompanhamento manual, piquete a piquete, sujeito a erro. Um programa nacional de intensificação combinou a subdivisão em piquetes e o ajuste da lotação a outras práticas, como adubação, correção do solo, suplementação e treinamento das equipes. Monitoradas por sete anos, as propriedades de referência multiplicaram por cinco a produtividade e mais que dobraram a capacidade de suporte animal (Grise *et al.*, 2025, p. 1, 3). Este trabalho parte da premissa de que o registro de ocupação e descanso de cada piquete é o insumo de que esse ajuste depende.

Para acompanhar esses tempos, há valores de referência na literatura. A unidade animal (UA) equivale a 450 kg de peso vivo (Martha Júnior et al., 2003, p. 4). Para as forrageiras tropicais na estação chuvosa, os períodos de descanso sugeridos vão de 20 a 45 dias, conforme a espécie (Martha Júnior et al., 2003, p. 3). Ocupações curtas, de menos de dez dias, são indicadas para pastagens muito adubadas (Martha Júnior et al., 2003, p. 4), e a Embrapa sugere até 7 dias de ocupação no pastejo rotacionado de gado de corte (Santos; Corrêa; Balsalobre, 2005, p. 12). Quanto à lotação, os valores estimados para distintos graus de intensificação do uso da terra vão de 1,18 a 6,90 UA/ha (Martha Júnior et al., 2003, p. 5).

2.3. Transformação Digital no Campo e Barreiras

Uma revisão sistemática de cinquenta artigos sobre desenvolvimento de software para o setor agropecuário identificou diversos obstáculos à agricultura digital entre agricultores familiares: conectividade, acesso à tecnologia e mão de obra qualificada (Gonçalves; Schlindwein, 2025, p. 58-59). Para o pequeno e o médio produtor, o custo de adquirir a tecnologia e a falta de preparo para usá-la também estão entre os entraves à adoção de Tecnologias de Informação e Comunicação (TIC) (Chacon; Marinho; Alves, 2023).

Mesmo em operações de maior porte, o tamanho do rebanho e o nível de instrução de quem administra a fazenda influenciam a decisão de adotar tecnologia digital. Um estudo com 185 fazendas brasileiras de confinamento comparou produtores que não usam software, os que informatizaram a gestão e os que

também automatizaram parte do manejo com sensores. Rebanhos maiores e responsáveis com ensino superior concentram-se nos grupos que mais usam software (Eusébio *et al.*, 2023, p. 1, 5). Se esse padrão aparece mesmo entre quem já dispõe de capital para investir, é plausível que a barreira seja maior nas pequenas propriedades familiares que este trabalho tem como público-alvo.

2.4. Sincronização de Dados em Sistemas de Gestão Pecuária

Trabalhos que já levaram a gestão pecuária ao celular do produtor seguem dois caminhos. O de Silva guarda os dados no celular e só precisa de conexão para autenticar o usuário (Silva, 2022). Os de Chacon e de Costa dependem de um servidor (Chacon; Marinho; Alves, 2023; Costa *et al.*, 2024). Nenhum deles trata a rede como algo dispensável por completo: autenticar o usuário ou sincronizar com um servidor central exige conexão em algum ponto do fluxo. O aplicativo deste trabalho tem o mesmo limite no login.

Um sistema de apoio ao pastejo rotacionado funciona sem rede e transmite os dados quando encontra conexão, porque o sinal nem sempre cobre toda a propriedade (Freitas *et al.*, 2024, p. 5). Nos sistemas de Chacon e de Costa, a conexão com o servidor é o caminho principal. Este trabalho inverte essa prioridade: a arquitetura de software adotada, apresentada na Seção 4.3, tem a operação local como caminho principal e a sincronização como reforço em segundo plano. Essa inversão coincide com o princípio dos *local-first software*, apresentado na Seção 2.5.

2.5. Arquiteturas *Offline-first* e Padrões de Projeto em Aplicações Móveis

Uma aplicação que precisa funcionar sem rede depende, antes de tudo, de decidir onde fica a cópia principal dos dados. No princípio dos *local-first software*, essa cópia reside no próprio dispositivo: o usuário lê e grava a qualquer momento, e a sincronização com os demais dispositivos ocorre depois, quando há conexão. Entre as razões, os autores incluem a conectividade limitada em regiões rurais (Kleppmann *et al.*, 2019, p. 156).

Resta garantir que o registro local e o envio ao servidor não se desencontrem. Em serviços, o problema equivalente é atualizar o banco de dados e publicar uma mensagem: se o processo falhar entre as duas operações, o sistema fica inconsistente. No padrão *Transactional Outbox*, a alteração dos dados e o registro da mensagem pendente ocorrem na mesma transação local, e a mensagem fica em uma tabela do próprio banco. A entrega cabe a um componente à parte, que, na forma mais simples, consulta a tabela de tempos em tempos (Richardson, 2019, p. 97-98). Como uma falha pode fazer a mesma mensagem chegar mais de uma vez, o receptor precisa tolerar repetições, seja tornando as operações idempotentes, seja ignorando os identificadores já vistos (Richardson, 2019, p. 95-96).

O restante do aplicativo também não deve depender da origem dos dados. Para isso, Hieatt e Mee, no catálogo de Fowler (2003, p. 322), descrevem o padrão *Repository*. Ele isola o domínio do acesso aos dados e expõe uma interface semelhante à de uma coleção. A mesma interface admite estratégias diferentes de busca, como objetos já carregados ou uma fonte externa, sem mudar o código que a utiliza (Fowler, 2003, p. 325). Esse isolamento segue a regra de dependência da Arquitetura Limpa: o código das camadas externas, como banco de dados e interface, depende das regras de negócio, e nunca o contrário (Martin, 2018, cap. 22).

No processo de desenvolvimento, o modelo de ramificação proposto por Driessen (2010) mantém duas ramificações permanentes: uma reflete o estado pronto para produção, e a outra reúne as mudanças

já entregues para a próxima versão. Ramificações de apoio, de vida curta, cuidam de funcionalidades, versões e correções urgentes.

Como o artefato não é avaliado com usuários, este trabalho deriva desse referencial cinco critérios, aferidos na Seção 3.4. O primeiro é a integridade dos dados sem conexão, sem perda de registros. O segundo é a consistência da sincronização: toda mutação criada deve chegar ao servidor ou permanecer na fila, à vista do produtor, sem desaparecer nem deixar registro órfão no servidor. A convergência sob edição concorrente não faz parte do critério. Esses dois vêm do princípio *local-first* e do padrão *Outbox*. O terceiro é a cobertura dos requisitos, inclusive do cálculo de autonomia dos insumos (Seção 2.1). O quarto vem da teoria do pastejo (Seção 2.2): o aplicativo deve registrar a ocupação e o descanso de cada piquete, os dois tempos da Equação 1, e avisar quando passam das referências. O quinto é o diferencial frente aos trabalhos correlatos da Seção 2.4. Verificar esses critérios por testes e por experimentos em ambiente controlado, sem usuários, está entre os métodos de avaliação admitidos na DSR (Hevner *et al.*, 2004, p. 86), detalhada na Seção 3.

3. Metodologia

3.1. Caracterização da Pesquisa

Esta pesquisa é de natureza aplicada, com abordagem predominantemente quantitativa, apoiada por uma conversa qualitativa com um produtor, e objetivo exploratório e descritivo. Quanto aos procedimentos, combina a pesquisa bibliográfica com a *Design Science Research* (DSR). Esse método operacionaliza a construção de conhecimento a partir de artefatos projetados para resolver problemas, avaliados e comunicados de forma rigorosa (Lacerda *et al.*, 2013, p. 744). A condução da DSR percorre cinco etapas: conscientização do problema, sugestão de alternativas, desenvolvimento do artefato, avaliação e conclusão (Lacerda *et al.*, 2013, p. 749-751). Na avaliação, confere-se o desempenho do artefato no contexto de uso previsto, diante dos objetivos que motivaram sua construção (Lacerda *et al.*, 2013, p. 750).

Entre os métodos de avaliação admitidos na DSR estão os testes funcionais, que acionam o artefato por suas interfaces em busca de falhas, e os experimentos em ambiente controlado (Hevner *et al.*, 2004, p. 86). Este trabalho adota esses dois métodos e não realiza avaliação com usuários finais. A escolha decorre do recorte do trabalho, que busca verificar a viabilidade técnica da arquitetura, e do prazo do curso, que não comporta um período de uso real em uma propriedade. A utilidade do aplicativo em uso real fica, portanto, como trabalho futuro.

3.2. Unidade de Análise e Observação

A unidade de análise é o próprio artefato, formado pelo aplicativo móvel e pela API que o sincroniza. Ele é analisado frente aos requisitos de gestão de uma pequena propriedade de bovinocultura de corte com conectividade restrita. As unidades de observação são os casos de teste automatizados e as mutações geradas nos cenários de desconexão, com os vereditos devolvidos pelo servidor.

3.3. Técnica de Coleta de Dados

Os dados vieram de três fontes. A primeira foi a pesquisa bibliográfica, feita principalmente no Google Acadêmico e complementada por repositórios institucionais, como os da Embrapa, do IBGE e da Sociedade Brasileira de Computação, e pelos sites dos próprios periódicos. A busca não foi sistemática, pois seu fim era fundamentar o problema e as escolhas do projeto, e não mapear toda a produção sobre o tema. A segunda foi o levantamento de requisitos descrito na Seção 3.5.1, que serviu apenas para definir o escopo do sistema. Recorreu-se a um único produtor, sem protocolo formal, porque a conversa delimitava o escopo e não gerava dados de pesquisa. A terceira são os registros gerados pela execução dos testes automatizados, no ambiente de desenvolvimento, e dos cenários controlados de desconexão, em aparelho físico.

Os cenários verificam o aplicativo com a rede desligada e religada, em um aparelho físico (Galaxy A22, com Android 13). Um roteiro escrito em Dart, instalado no aparelho, executou os quatro cenários três vezes cada, com a fila de saída e o gerenciador de sincronização reais. A API rodou na máquina de desenvolvimento, ligada ao aparelho por cabo USB. O Wi-Fi e os dados móveis foram desligados e religados por comando, e o monitor de conectividade do aplicativo detectou a queda e a volta da rede. O segundo cliente dos cenários C e D foi simulado por chamadas diretas à API. O produtor, nesses dois cenários, foi representado pelo roteiro, que chama os repositórios do aplicativo, e não pela tela de pendências.

3.4. Técnica de Análise dos Dados

A análise segue os critérios definidos na Seção 2.5. Cada requisito é confrontado com o caso de teste que o verifica e classificado como atendido ou não atendido (critério 3). Os critérios 1 e 2 são aferidos pelos cenários de desconexão e pelos testes da fila de envio, e o critério 4, pelos testes das regras de descanso, ocupação e lotação. Os testes automatizados cobrem os dois lados do sistema, e ambas as suítes rodam em um *pipeline* de integração contínua a cada *merge request*. No aplicativo, a suíte executada com `flutter test` verifica as regras de cada módulo e o mecanismo de sincronização. Na API, a suíte Vitest exercita os *endpoints* contra um banco PostgreSQL real.

Os cenários de desconexão registram, para cada situação simulada, as mutações criadas e as sincronizadas, as perdas, as divergências provocadas e o tempo até a fila ficar sem pendências. Eles foram executados em duas rodadas: a primeira revelou falhas no aplicativo, que foram corrigidas, e os quatro cenários foram repetidos. Por fim, o critério 5 é aferido por comparação analítica com os trabalhos correlatos da Seção 2.4: cada publicação é examinada nos aspectos centrais do recorte, como lotes, pastagens, insumos e operação sem conexão, com base no que ela descreve, e o resultado é organizado em quadro (Seção 5.4). Os números da segunda rodada são apresentados na Seção 5.

3.5. Procedimentos de Desenvolvimento do Artefato

3.5.1. Levantamento de Requisitos

Os requisitos funcionais do aplicativo *Pasto Livre* e da API partiram de duas fontes. A primeira foi a vivência do autor no meio rural, com contato prévio com a pecuária de corte. A segunda foi uma conversa com um produtor de uma pequena propriedade de bovinocultura de corte que adota o pastejo rotacionado.

A conversa serviu apenas para levantar requisitos e definir o escopo, sem coleta de dados de pesquisa, e o produtor não é identificado. Dela vieram três prioridades: organizar o rebanho por lotes, registrar o histórico de manejo das pastagens e controlar o estoque de insumos. Essas prioridades atendem ao primeiro objetivo específico do trabalho (Seção 1).

3.5.2. Modelo de Processo

O desenvolvimento foi organizado em Kanban. O Scrum estrutura o trabalho em *sprints* de até um mês, conduzidos por um time com papéis definidos (Schwaber; Sutherland, 2020, p. 6, 8). O custo desses papéis tende a superar o benefício para um desenvolvedor único. O Kanban otimiza o fluxo por meio de um sistema visual e puxado, em que um item só começa quando há capacidade para ele (Vacanti; Coleman, 2020, p. 4, 6). Esse fluxo acompanha melhor a disponibilidade irregular de quem concilia o desenvolvimento com outras obrigações.

O *backlog* de funcionalidades é mantido como *issues* no GitLab. Cada *issue* recebe um rótulo de fase (*workflow::*) que a posiciona no quadro Kanban do grupo. O quadro tem seis fases, de *Backlog* a *Closed*, passando por *Opened*, *Doing*, *Review* e *QA*. Os rótulos *prio::* e *size::* registram a prioridade e o tamanho estimado da tarefa, e os três permitem ver o que está pronto, em andamento ou por fazer.

3.5.3. Tecnologias Adotadas

O aplicativo móvel é construído em Flutter, *framework* de código aberto mantido pelo Google que gera aplicações para mais de uma plataforma a partir de uma única base de código (Google, 2026; Marques et al., 2022, p. 5). A entrega atual restringe-se ao Android, mas a escolha evita reescrever a base do zero caso a expansão para iOS, já prevista, aconteça. Duas bases de código nativas, uma por plataforma, dobrariam o esforço de manutenção, inviável para um desenvolvedor único. A gerência de estado usa a biblioteca `flutter_bloc`, e a persistência local usa `sqflite`, SQLite embarcado que dispensa servidor de banco no próprio dispositivo. Essa persistência sustenta o uso diário sem conexão, coerente com a arquitetura *offline-first* discutida na Seção 2.5; a exceção é o login, que exige conexão. O funcionamento sem conexão e a sincronização com a nuvem estão entre as demandas próprias das aplicações móveis (Ferreira; Ferreira; Grande, 2022, p. 6).

A API é construída em Node.js com Express 5, *framework* minimalista que não impõe uma estrutura de projeto (Express, 2026). A escolha considerou a familiaridade do autor com a ferramenta, o que reduz o tempo de aprendizado em um projeto conduzido por uma só pessoa. O Prisma[4] faz o papel de *Object-Relational Mapping* (ORM) sobre PostgreSQL. Ele gera automaticamente, a partir do *schema* do banco, um cliente de consultas com tipagem, o que reduz erros manuais de SQL. A autenticação usa `better-auth`[5], *framework* de autenticação que já traz recursos prontos, como o gerenciamento de sessões. Assim, evita-se reimplementar do zero um mecanismo sensível à segurança.

No apoio ao processo, as telas foram prototipadas no Figma[6], o que tende a evitar retrabalho de có-

[4] PRISMA. *Prisma ORM documentation*. Disponível em: https://www.prisma.io/docs. Acesso em: 2 out. 2026.

[5] BETTER AUTH. *Better Auth documentation*. Disponível em: https://www.better-auth.com/docs. Acesso em: 2 out. 2026.

[6] FIGMA. *Figma*. Disponível em: https://www.figma.com/. Acesso em: 2 out. 2026.

digo. A modelagem do banco relacional partiu de diagramas no DBDiagram[7], posteriormente traduzidos para o *schema* do Prisma.

4. Desenvolvimento

4.1. Fluxo de Desenvolvimento

O Kanban e o rótulo *workflow::* organizam o que fazer e em que ordem, conforme descrito na Seção 3.5.2. A construção de cada funcionalidade, por sua vez, segue o Gitflow (Driessen, 2010): cada *issue* abre uma *branch* própria a partir de `develop`, integrada de volta por *merge request* depois de revisada. A *branch* `main` recebe apenas versões de *release*, nunca *commit* direto. Mobile e API são repositórios Git separados, com esse mesmo fluxo replicado em ambos. Quando uma funcionalidade depende dos dois lados, as *issues* ficam referenciadas entre si, o que mantém rastreável a mudança de contrato da API e o consumo dela pelo aplicativo.

4.2. Módulos Funcionais

O núcleo do aplicativo divide-se em três módulos, que correspondem às prioridades levantadas com o produtor (Seção 3.5.1): lotes, pastagens e insumos, todos vinculados a uma propriedade escolhida pelo usuário. Em torno deles estão o cadastro da conta e das propriedades, o tutorial de primeiro acesso, a tela inicial com resumo e alertas e o relatório de resultado do período. Completam o conjunto as notificações, o registro fotográfico, a exportação dos dados e o aceite da política de privacidade. O Quadro 1 resume o que cada módulo central entrega e a regra de negócio que o distingue de um cadastro simples.

Quadro 1. Módulos funcionais do aplicativo e suas regras de negócio

Módulo	**Funcionalidades**	**Regra implementada**
Pastagens	Cadastro; status; registro de manejos	Status *Vazio*, *Ocupado* ou *Descanso*; contador de dias de descanso frente ao alvo do pasto; aviso de ocupação acima de 7 dias; taxa de lotação da propriedade em unidade animal por hectare (UA/ha), com aviso acima de 2,0
Lotes	Cadastro; movimentação entre pastos; manejo sanitário; saída por venda, morte, abate ou outro motivo	A movimentação atualiza o status do pasto de origem e o do destino. Entrar em pasto em descanso ou ocupado pede confirmação. O manejo registra vacinação, vermifugação e pesagem. A venda guarda o preço da arroba e o valor total
Insumos	Cadastro; entradas e saídas; consumo diário; gasto por lote	Autonomia em dias (saldo dividido pelo consumo diário); aviso de estoque esgotado ou baixo

Fonte: Elaborado pelo autor (2026), a partir do código do aplicativo.

O módulo de pastagens trabalha com três parâmetros, fundamentados na Seção 2.2. A UA é calculada com 450 kg de peso vivo. Quando o lote não informa o peso médio, o aplicativo o estima pelo sistema de produção (400 kg na cria, 300 kg na recria e 450 kg na engorda) e sinaliza o valor como estimado. O alvo de descanso é definido para cada pasto, em três níveis de prioridade. O primeiro é o valor que o produtor informa, de 1 a 365 dias. Sem ele, vale o padrão da forrageira escolhida. O aplicativo traz um catálogo de nove forrageiras, mantido no servidor e sincronizado com o aparelho. Cada uma tem um padrão de

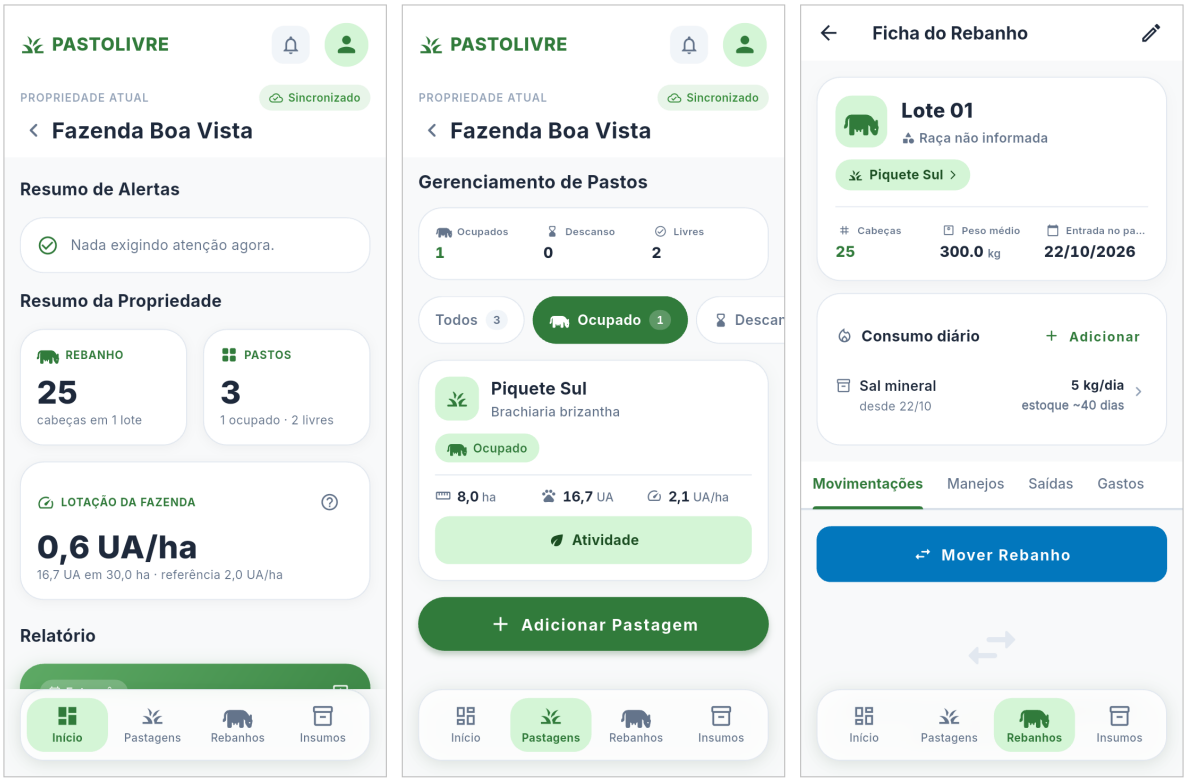
[7] HOLISTICS. *dbdiagram.io*. Disponível em: https://dbdiagram.io/. Acesso em: 2 out. 2026.

25 a 38 dias, que corresponde à média da faixa sugerida para o período das águas (Martha Júnior et al., 2003, p. 3). Sem forrageira, vale a referência geral de 30 dias, dentro das faixas apresentadas na Seção 2.2. O padrão do catálogo não considera solo, região nem época do ano.

A ocupação de 7 dias e a lotação de 2,0 UA/ha são referências fixas, escolhidas dentro dos valores apresentados na Seção 2.2. A lotação é calculada para a propriedade inteira, somando as UA dos lotes ativos e dividindo pela área dos pastos com área cadastrada. No piquete, a densidade é apenas informada. A ocupação de 7 dias também coincide com o exemplo numérico de Braga *et al.* (2020, p. 7). Descanso e ocupação são os dois tempos da Equação 1. Como o descanso varia com a espécie e com a época do ano (Seção 2.2), o aplicativo trata os três valores como alvos visuais: avisam, mas não impedem a movimentação, e a decisão fica com o produtor. O ajuste à época do ano e às condições da propriedade é trabalho futuro.

A integração entre lotes e pastagens é o ponto central do conjunto. Ao movimentar um lote, o aplicativo atualiza o status dos pastos envolvidos na mesma transação local, sem ação adicional do usuário. A tela inicial usa esses dados para alertar sobre lotes sem pasto, lotes há mais de 7 dias no mesmo piquete, pastos que já cumpriram o período de rebrota e lotação da propriedade acima da referência. No módulo de insumos, o saldo projetado é o saldo em estoque descontado o consumo estimado ainda não lançado. Como no pastejo, o resultado é um indicador de apoio: o aplicativo sinaliza, e a decisão continua com o pecuarista.

Figura 2. Telas dos módulos: resumo da propriedade, pastagens e ficha do lote



(a) (b) (c)

Fonte: Elaborado pelo autor (2026), capturas de tela no aparelho físico, com dados de uma propriedade de demonstração.

A Figura 2 mostra três telas, capturadas no aparelho físico com dados de uma propriedade de de-

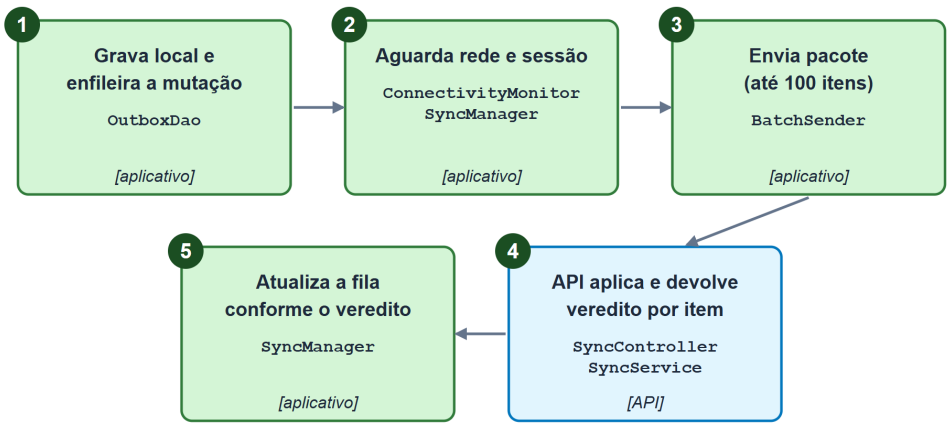
monstração. A tela inicial (a) mostra a lotação da fazenda, de 0,6 UA/ha, ao lado da referência de 2,0. Em (b), o Piquete Sul aparece ocupado, com 16,7 UA em 8 ha, e a densidade de 2,1 UA/ha é apenas informada. Em (c), o consumo de sal mineral, de 5 kg/dia, dá ao estoque uma autonomia de cerca de 40 dias.

4.3. Arquitetura do Sistema

Os módulos descritos operam sobre o banco local e, salvo no login, comunicam-se com a API apenas pela sincronização, que sustenta a operação *offline-first* descrita na Seção 2.4. O envio das mutações e a leitura das alterações seguem caminhos distintos, descritos a seguir. A escrita nunca chama a API direto. Ao criar, editar ou excluir um registro, o aplicativo grava a mudança no banco local e enfileira a mutação pendente em uma fila de saída (*outbox*), na mesma transação do SQLite, conforme o padrão *Transactional Outbox* (Richardson, 2019, p. 97-98). O gerenciador de sincronização envia a fila quando há rede e sessão autenticada, e a retoma quando uma espera programada vence. Cada rodada envia até cem itens para a API e se repete até esvaziar o que está liberado. Cada item recebe um veredito individual: aceito, recusado, bloqueado por depender de outra mutação ainda não confirmada ou em conflito, quando o registro mudou no servidor depois que o aparelho o baixou.

A API responde sempre com HTTP 200 para o envio inteiro, mesmo quando recusa uma mutação específica. Para um *POST*, o código 200 indica o estado da ação ou os resultados obtidos com ela (Fielding; Nottingham; Reschke, 2022, seç. 15.3.1), e o veredito de cada item segue no corpo da resposta. O status HTTP descreve o transporte do lote, não o resultado de cada item; um erro 4xx faria o cliente descartar as demais mutações do mesmo envio. Recebido o veredito, o aplicativo remove da fila a mutação aceita. A recusa recuperável, como uma falha momentânea do servidor, devolve a mutação a pendente, com espera, até cinco tentativas. A recusa definitiva, como um nome de pasto repetido, deixa a mutação em falha até o produtor reenviá-la, corrigi-la ou descartá-la. Descartar um cadastro recusado desfaz o efeito dele no aparelho: o registro e as mutações que dependem dele saem do banco local, e um lote movimentado para esse pasto volta ao pasto de origem. A mutação bloqueada não é tentada, e a em conflito espera a escolha do produtor, descrita adiante.

Figura 3. Ciclo de sincronização de uma mutação, do registro local ao veredito da API



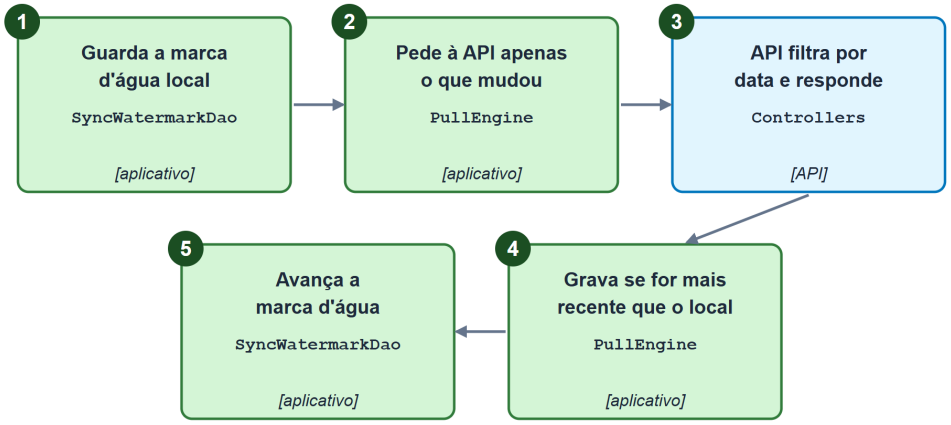
Fonte: Elaborado pelo autor (2026).

A Figura 3 resume esse ciclo em cinco passos. O quarto passo ocorre na API, e os demais, no aplicativo, com a classe responsável indicada em cada caixa.

O reenvio é seguro. Cada mutação recebe no aparelho um identificador único, e o servidor registra, na mesma transação do efeito, os identificadores já aplicados. Se o mesmo identificador chegar de novo, por exemplo porque a resposta se perdeu, o servidor devolve o resultado arquivado, por 30 dias, sem repetir o efeito. A fila fica em uma tabela do SQLite e sobrevive ao fechamento do aplicativo. A exclusão local é lógica: o registro permanece marcado como inativo, para que a leitura não o restabeleça.

A leitura segue o caminho inverso e usa uma marca d'água: para cada tipo de dado e cada recorte de consulta, o aplicativo guarda a data da última atualização já recebida. A cada sincronização, ele solicita à API apenas os registros alterados depois dessa marca, em vez de recarregar a base inteira. A API filtra os registros por essa data e devolve só o que mudou. A tela sempre mostra o que já está salvo localmente, mesmo durante essa espera. Gravado o dado, a marca d'água avança, o que evita repetir dados já lidos.

Figura 4. Ciclo de leitura por marca d'água, do pedido das alterações à gravação local



Fonte: Elaborado pelo autor (2026).

A Figura 4 resume esse caminho em cinco passos, da marca d'água guardada no aparelho até o avanço dela depois da gravação.

A resolução de conflitos tem duas regras distintas. Na leitura, o registro do servidor só substitui o local quando seu carimbo de atualização, atribuído pelo servidor, é posterior ao da cópia local, que o aplicativo carimba com o relógio do aparelho. Registros citados por qualquer mutação ainda na fila, inclusive as que falharam, nunca são sobrescritos. Em consequência, um relógio adiantado pode impedir que edições posteriores feitas em outro aparelho substituam a cópia local.

No envio, a edição concorrente é detectada por versão do registro. Oito tipos de registro editável têm um número de versão, que nasce em 1 e sobe a cada edição do usuário. O aplicativo guarda a versão junto com o registro e, ao enfileirar uma edição, anexa a versão em que ela se baseou. Se o registro tiver mudado no servidor desde então, o servidor não aplica a edição e devolve o veredito de conflito, e as mutações que dependem dela voltam bloqueadas. Duas edições seguidas do mesmo registro na fila ficam encadeadas, e a segunda parte da versão que a primeira produz. Em conflito, o produtor escolhe manter a edição do aparelho, reenviada por cima da versão do servidor, ou usar a do servidor, o que descarta a edição local. Edições sem versão de base, de aparelhos antigos, seguem a regra da última escrita vence (*last-write-wins*). A comparação vale para o registro inteiro, e não por campo, e a exclusão

não é verificada.

4.4. Escrita *Offline* na Prática

O trecho a seguir, do repositório de pastagens do aplicativo, mostra o primeiro passo do ciclo da Figura 3 no código. Ao criar uma pastagem, o registro local e a mutação pendente são gravados na mesma transação do SQLite (Quadro 2). O gerenciador de sincronização é avisado apenas depois da confirmação.

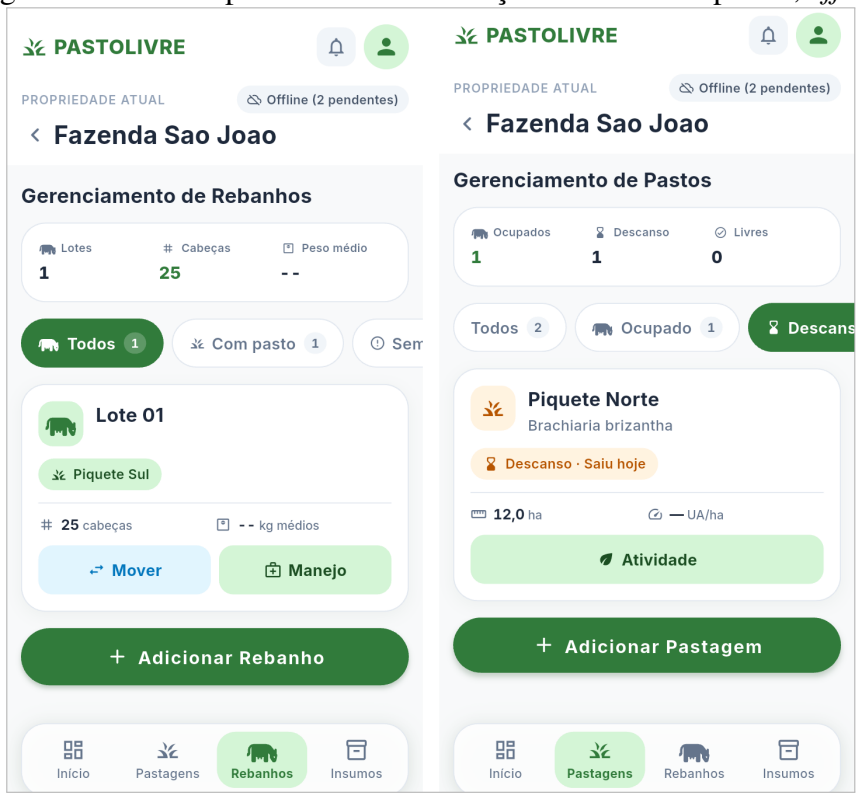
Quadro 2. Escrita local e enfileiramento da mutação na mesma transação

```
final db = await _dbHelper.database;
await db.transaction((txn) async {
  await _local.upsert(localModel, txn);
  await _outbox.enqueue(
    PendingMutation(
      id: _uuid.v4(),
      entity: SyncEntity.pastos,
      action: SyncAction.create,
      entityId: newId,
      data: dados,
      dependsOn: dependeDe,
      createdAt: agora,
    ),
    txn,
  );
});
unawaited(_syncManager.notifyChange());
```

Fonte: Elaborado pelo autor (2026), trecho de `PastoRepositoryImpl.createPasto`.

Se a transação falhar, nem o registro local nem a mutação existem, e nenhuma mutação fica órfã na fila. Se ela for concluída, a tela já mostra a pastagem criada, e o envio só é tentado se houver conexão.

Figura 5. Telas do aplicativo: movimentação de lote entre pastos, *offline*



Fonte: Elaborado pelo autor (2026), captura de tela no aparelho físico.

O estado dessa fila aparece ao usuário no distintivo de sincronização da barra superior. A Figura 5 registra, no aparelho físico e sem conexão, duas mutações pendentes: a criação do pasto “Piquete Sul”, pelo método do Quadro 2, e a movimentação do “Lote 01” para esse pasto. À esquerda, o módulo de rebanhos mostra o lote vinculado ao novo pasto. À direita, o módulo de pastagens mostra o “Piquete Norte”, de origem, em *Descanso* assim que o lote saiu, sem ação do usuário nem envio ao servidor. O módulo de insumos grava pelo mesmo caminho, mas não aparece em captura.

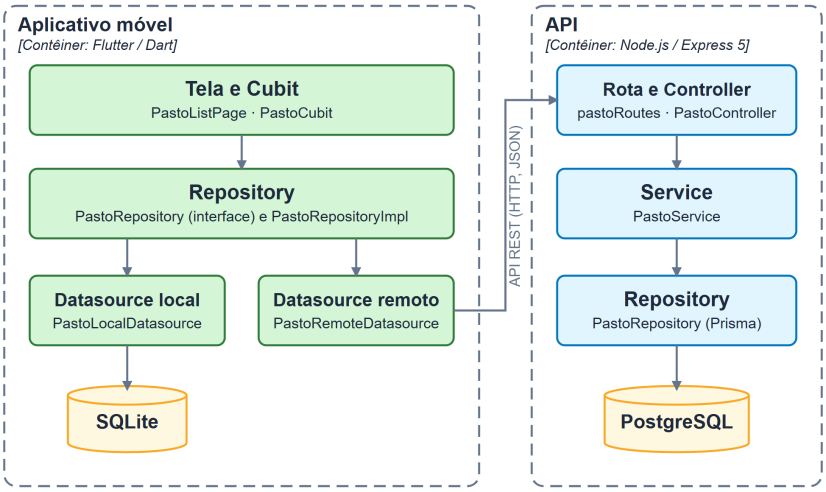
4.5. Padrões de Projeto

Além do *Outbox*, já apresentado, o sistema adota outros padrões de projeto. Tanto o aplicativo quanto a API seguem o padrão *Repository* (Fowler, 2003, p. 322), que isola o acesso a dados (SQLite ou Prisma) atrás de uma classe dedicada, com responsabilidade única sobre a fonte de dado. A tela (*page*) do aplicativo não acessa dado algum diretamente: delega ao *Cubit*, que aplica a regra de apresentação e emite estados observados pela interface, conforme o padrão Bloc/Cubit (Bloc, 2026). Ao depender da interface `PastoRepository`, e não da implementação concreta, o *Cubit* aplica o princípio da inversão de dependências: o código se apoia em contratos genéricos, e não em implementações específicas (Ferreira; Ferreira; Grande, 2022, p. 3). A classe `PastoRepositoryImpl` decide entre as duas fontes de dado: o *datasource* local, que fala com o SQLite, e o remoto, que fala com a API REST via `dio`. É essa decisão, gravar sempre no banco local e enfileirar a mutação para envio posterior, que sustenta o mecanismo *Outbox* descrito na Seção 4.3.

Do lado do servidor, a requisição chega pela rota do Express e desce por três camadas encadeadas: o *controller* valida a entrada, o *service* aplica a regra de negócio e o *repository* fala com o banco via

Prisma. Essa divisão segue o princípio comum às arquiteturas reunidas sob a Arquitetura Limpa: separar as regras de negócio dos detalhes externos, como o banco de dados e a interface (Martin, 2018, cap. 22). Diferentemente do aplicativo, essas classes da API são concretas e encadeadas por instanciação direta: a rota cria o *controller*, que cria o *service*, o qual usa instâncias únicas dos *repositories*, sem interface entre elas.

Figura 6. Diagrama de componentes do aplicativo móvel e da API



Fonte: Elaborado pelo autor (2026), na notação do C4 Model (Brown, 2026).

A Figura 6 organiza as duas lado a lado em um diagrama de componentes do C4 Model, com a seta ao centro marcando onde uma cadeia aciona a outra.

A resolução dessas dependências, no aplicativo, segue o padrão *Service Locator*, em que um objeto sabe obter os serviços de que a aplicação precisa (Fowler, 2004). O `get_it` entrega a instância registrada, e o `injectable` gera esse registro a partir de anotações no código, como `@LazySingleton`, que torna o repositório uma instância única do aplicativo.

4.6. Uso de Inteligência Artificial

O desenvolvimento dos dois repositórios foi assistido pelo Claude Code (Anthropic, 2026), ferramenta de linha de comando que roda modelos da família Claude (Anthropic) com acesso ao sistema de arquivos e ao terminal. O processo segue convenções documentadas em cada repositório. Uma *skill* de *scaffolding* replica os padrões arquiteturais já estabelecidos ao criar um domínio novo, e um agente de revisão dedicado confere cada alteração contra essas convenções antes da abertura de uma *merge request*. Conforme essas convenções, nenhuma alteração segue sem validação: `flutter analyze` e `flutter test` no aplicativo e a suíte de *endpoints* na API rodam localmente antes do *push* e voltam a rodar no *pipeline* de cada *merge request*.

A mesma ferramenta auxiliou na redação, na revisão e na conferência das citações do artigo contra os PDFs do acervo. O autor revisou o código e o texto produzidos com esse apoio e responde integralmente pelo conteúdo do trabalho. Não foi localizada norma do Instituto Federal de Rondônia (IFRO) sobre o uso de inteligência artificial em trabalhos de conclusão de curso, e este uso é declarado por transparência.

4.7. Publicação e Infraestrutura

A publicação da API é automatizada por um *pipeline* de integração contínua no GitLab[8], que implanta o serviço em um cluster Kubernetes[9]. A suíte de testes roda contra um PostgreSQL real em toda *merge request*. Só na `main` o *pipeline* constrói a imagem de contêiner com a ferramenta Kaniko, atualiza o *deployment* e confirma a saúde dos novos *pods*, e cada imagem leva o *hash* curto do *commit* que a gerou. A API roda em um cluster leve (K3s) na Oracle Cloud, e o aplicativo a alcança por um túnel do Cloudflare, sem porta aberta no servidor. O aplicativo tem um *pipeline* próprio, que roda `flutter analyze` e `flutter test` em cada *merge request*, mas não gera nem distribui o aplicativo.

5. Resultados e Discussão

5.1. Requisitos Atendidos

O plano de teste do aplicativo define 17 requisitos funcionais (RF) e 11 não funcionais (RNF). Cada um se liga a cenários com identificador estável, repetido no nome do teste que o verifica. O Quadro 3 mostra, para cada RF, quantos testes o verificam e o resultado. Um mesmo cenário pode verificar mais de um requisito, e por isso os totais não se somam.

Quadro 3. Requisitos funcionais do aplicativo e resultado dos testes

Código	**Requisito**	**Testes**	**Resultado**
RF-001	Autenticar e manter a conta	24	Atendido
RF-002	Apresentar o tutorial de primeiro acesso	7	Atendido
RF-003	Manter propriedades	39	Atendido
RF-004	Manter pastagens	28	Atendido
RF-005	Controlar o descanso do pasto	20	Atendido
RF-006	Calcular a lotação	24	Atendido
RF-007	Manter rebanhos e movimentar lotes	40	Atendido
RF-008	Registrar a compra do lote	11	Atendido
RF-009	Registrar a saída de animais	38	Atendido
RF-010	Controlar o estoque de insumos	100	Atendido
RF-011	Lançar entradas e saídas de insumo	71	Atendido
RF-012	Registrar manejos com insumos	18	Atendido
RF-013	Apurar gastos e relatório	29	Atendido
RF-014	Resumir a propriedade na tela inicial	14	Atendido
RF-015	Notificar o produtor	29	Atendido
RF-016	Fotografar as atividades	40	Atendido
RF-017	Exportar os dados e aceitar a política de privacidade	11	Atendido

Fonte: Elaborado pelo autor (2026), a partir do plano de teste do aplicativo e do relatório da suíte.

Os 17 requisitos funcionais foram atendidos, pois todos os testes que os verificam foram aprovados. Os requisitos não funcionais, concentrados na sincronização, também foram atendidos: o envio sem perda tem 79 testes, a ordem e a dependência entre mutações 62 e a leitura que não sobrescreve o dado local

[8] GITLAB. *GitLab CI/CD*. Disponível em: https://docs.gitlab.com/ci/. Acesso em: 13 out. 2026.

[9] KUBERNETES. *Kubernetes documentation*. Disponível em: https://kubernetes.io/docs/. Acesso em: 13 out. 2026.

60. A rastreabilidade das regras (RNF-011) é verificada apenas pela convenção de nomes: cada um dos 663 testes da suíte carrega o identificador de um cenário, o que não prova a adequação das regras.

5.2. Cenários de Desconexão

Os cenários seguiram o desenho descrito na Seção 3.3: quatro situações, repetidas três vezes cada, em um aparelho físico.

Os cenários rodaram em duas rodadas. Na primeira, o aplicativo ainda falhava: em D, sobrescreveu a edição mais recente do outro cliente e, em C, manteve no aparelho o efeito da mutação recusada. Corrigiu-se o aplicativo e os quatro cenários foram repetidos. Os resultados abaixo são os da segunda rodada. A edição concorrente não faz parte do critério 2 (Seção 2.5) e foi examinada à parte, depois da primeira.

O Quadro 4 traz as mutações criadas, as perdas, as divergências provocadas e o tempo até a fila ficar sem pendências, contado da detecção da rede. Perda é a mutação que deixa a fila sem ser aceita pelo servidor nem descartada pelo produtor. Divergência provocada é o conflito criado de propósito entre o aparelho e o segundo cliente.

No cenário A, sem rede, criam-se 30 registros por repetição (10 pastos, 10 lotes e 10 manejos), e todos chegaram ao servidor, sem perda. No B, cria-se um pasto e move-se um lote para ele. O pasto chegou antes da movimentação, na ordem causal declarada, e o lote terminou no pasto de destino nas três repetições.

No C, o servidor já tem um pasto de mesmo nome, criado por outro cliente, e o aparelho cria o seu e move um lote para ele. O servidor recusou o cadastro, e a movimentação, que dependia dele, ficou bloqueada sem ser tentada. O servidor não ficou com registro algum do aparelho, e o lote permaneceu no pasto de origem. No aparelho, enquanto o produtor não decidia, o lote seguia exibido no pasto recusado, e as duas mutações ficavam na fila, à vista dele. Ao descartar o cadastro recusado, o aplicativo removeu do banco local o pasto e a movimentação, e o lote voltou ao pasto de origem nas três repetições.

No D, o aparelho edita sem rede o nome de um pasto, e o outro cliente edita o mesmo pasto no servidor, depois. Nas três repetições, o servidor não aplicou a edição do aparelho: devolveu o conflito, com a versão de base 1 contra a versão 2 do servidor, e manteve a edição do outro cliente. A edição do aparelho ficou na fila, à espera do produtor. Quando ele escolheu manter a edição do aparelho (repetições 1 e 3), o aplicativo a reenviou, e o servidor a aplicou e subiu a versão para 3. Quando escolheu a do servidor (repetição 2), a edição local foi descartada, e o nome do servidor passou ao aparelho.

O tempo até a fila sem pendências foi de 1,4 s no cenário A e de 0,4 s no B, ou cerca de 5 s no A se contada a reassociação do Wi-Fi. Em C e D o tempo não se aplica, porque a fila só se esvazia depois da escolha do produtor. Os cenários não exercitaram a queda da rede durante o envio, a perda da resposta do servidor nem o fechamento do aplicativo com a fila pendente, casos cobertos só por testes automatizados isolados (Seção 5.3).

Quadro 4. Resultado dos cenários controlados de desconexão

Cenário	**Mutações criadas**	**Sincronizadas**	**Perdas**	**Divergências provocadas**	**Tempo até sincronizar**
A: criação em massa	30	30	0	0	1,4 s
B: mutações dependentes	2	2	0	0	0,4 s
C: recusa com dependente	2	0	0	1 (recusada)	n/a
D: edição concorrente	1	0 (1 em conflito)	0	1 (detectada)	n/a

Fonte: Elaborado pelo autor (2026), a partir da execução em aparelho físico. Em C e D, a divergência ocorreu entre o aparelho e um segundo cliente simulado, e o tempo não se aplica porque a fila aguarda o produtor.

5.3. Testes Automatizados

A suíte do aplicativo foi escrita com a biblioteca `flutter_test`, do próprio Flutter, e executada por `flutter test`. Ela tem 663 testes em 73 arquivos, todos aprovados. Eles verificam a regra de negócio sobre um SQLite em memória com o esquema real, obtido pelo pacote `sqflite_common_ffi`, e conferem o efeito no banco e na fila de envio, não só o retorno do método. Só a API, o Firebase e o armazenamento seguro são simulados, com a biblioteca `mocktail`. A cobertura de linhas é de 35,2%, valor baixo porque as telas, com 38% das linhas, têm 8,6%: a suíte protege regra, não aparência. Na camada de regra, a cobertura é de 72,9%, e chega a 92,6% na sincronização e a 100% na validação antes de gravar.

Para o critério 4, o descanso do pasto (padrão por forrageira e ajuste por pasto) tem 20 testes, a lotação e a ocupação do piquete 24 e os avisos da tela inicial 14. Entre eles, a ocupação é avisada acima de 7 dias e a lotação acima de 2,0 UA/ha é sinalizada como acima da referência, e ambas só avisam. No piquete, a densidade é apenas informada.

A suíte da API foi escrita com o Vitest e tem 1.050 testes em 81 arquivos, todos aprovados. Eles usam o Supertest para exercitar os *endpoints* por HTTP contra um PostgreSQL real, e o provedor V8 do Vitest mede a cobertura: 93,7% das linhas e 85,4% dos ramos. Os 118 testes de sincronização tratam do envio em lote, dos vereditos por mutação, das dependências, da idempotência e da detecção de conflito por versão. A suíte roda a cada *merge request* no *pipeline* da Seção 4.7.

Os testes das duas suítes verificam a implementação, e não a adequação das regras ao manejo real. O mesmo autor escreveu o aplicativo e os testes, e por isso a aprovação dos testes não substitui uma avaliação independente.

Em síntese, o critério 1 (integridade dos dados sem conexão) foi atendido nos cenários A e B e nos testes da fila. O critério 2 (consistência da sincronização) foi atendido: as mutações aceitas chegaram ao servidor, e a recusa ficou visível na fila e foi desfeita no aparelho pelo produtor. A edição concorrente, que o critério não cobre, foi detectada no cenário D. Os critérios 3 e 4 foram atendidos nos testes, e o critério 5 é discutido na Seção 5.4.

5.4. Comparação com os Trabalhos Correlatos

O Quadro 5 compara o aplicativo aos quatro trabalhos correlatos da Seção 2.4, nos aspectos que o recorte deste trabalho torna centrais. A comparação é analítica: baseia-se no que cada publicação des-

creve, e não na execução dos sistemas. Quando a publicação não trata de um aspecto, o quadro registra que ela não o descreve, o que não prova a ausência do recurso. Os quatro trabalhos vêm de busca não sistemática (Seção 3.3). Os sistemas comerciais citados na Introdução ficaram fora do quadro, porque não foram examinados neste trabalho.

Quadro 5. Comparação analítica entre o Pasto Livre e os trabalhos correlatos

Aspecto	**AgroTec**	**Costa et al.**	**Silva**	**Freitas et al.**	**Pasto Livre**
Plataforma	Ionic (Android e iOS)	Web (ReactJS)	Android nativo	Celular, computador e tablet	Flutter, Android
Operação sem conexão	Não descreve	Não descreve	Sim, após o login	Sim, projetada	Sim, exceto o login
Lotes	Sim	Não descreve	Sim	Não descreve	Sim
Pastagens	Cita o remanejamento entre pastos	Manejo de transferência de pastagem	Não	Rotação com cálculo de dias	Status, descanso, ocupação e lotação
Insumos	Não descreve	Não descreve	Não	Não descreve	Estoque, autonomia e gasto
Sincronização	Servidor, com conexão	Cliente-servidor	Só o login usa o servidor	Envia quando há conexão	Fila de saída, leitura por diferença e conflito por versão

Fonte: Elaborado pelo autor (2026), a partir de Chacon, Marinho e Alves (2023, p. 3-5), Costa *et al.* (2024, p. 2-3), Silva (2022, p. 19, 24-25, 28) e Freitas *et al.* (2024, p. 5-6).

Nenhum dos quatro trabalhos reúne lotes, pastagens e insumos. O AgroTec, com servidor PHP e banco MySQL, comunica-se com o servidor quando o aparelho tem internet (Chacon; Marinho; Alves, 2023, p. 4). O sistema de Costa *et al.* (2024, p. 2) também segue o modelo cliente-servidor, e a validação com o produtor estava pendente (Costa *et al.*, 2024, p. 3). O aplicativo de Silva (2022, p. 24-25) guarda os dados no aparelho e só precisa de conexão para autenticar o usuário. O sistema de Freitas *et al.* (2024, p. 5) funciona sem rede e transmite os dados assim que encontra conexão. É o mais próximo deste trabalho no uso sem rede, mas concentra-se na rotação dos piquetes e não descreve lotes, insumos nem o mecanismo de sincronização.

No uso sem conexão, o Pasto Livre e o aplicativo de Silva (2022) compartilham o mesmo limite, o login. O diferencial do Pasto Livre está na combinação. Ele reúne o controle de lotes, o histórico das pastagens rotacionadas e o estoque de insumos. Além disso, adota a operação sem conexão como regra, com uma fila de saída e uma leitura por diferença que preservam o dado local. Em contrapartida, Freitas *et al.* (2024, p. 6) fizeram um teste de cinco meses em uma fazenda e compararam a produção de forragem com a do ano anterior. Este trabalho não faz avaliação em uso real, e essa lacuna é tratada na seção seguinte.

5.5. Implicações, Limitações e Trabalhos Futuros

Do ponto de vista teórico, o trabalho adota, em um domínio específico, o princípio dos *local-first software* (Kleppmann *et al.*, 2019), sem propor um modelo novo de sincronização. A cópia principal do dado fica no aparelho, e a sincronização com o servidor ocorre depois, em segundo plano. Os resultados indicam que essa arquitetura é viável para a gestão pecuária de pequeno porte, no sentido de que as regras do aplicativo e da sincronização se mantiveram nos testes executados.

Os resultados também tensionam o princípio em um ponto prático. Com a cópia principal no aparelho, a edição concorrente precisa ser resolvida, e o mecanismo adotado detecta o conflito pela versão do registro inteiro, e não campo a campo. Essa granularidade deixa a escolha com o produtor, o que pressupõe conflitos raros, hipótese não medida neste trabalho. Quanto ao padrão *Outbox* (Richardson, 2019), a gravação local e a mutação pendente na mesma transação não deixaram mutação órfã, e o reenvio foi tratado pelos identificadores já aplicados no servidor, verificados nos testes automatizados.

Na teoria do pastejo, o trabalho não altera a Equação 1, mas trata o registro de ocupação e descanso de cada piquete como o insumo de que ela depende (Seção 2.2). O aplicativo mede os dois tempos da equação a partir das movimentações dos lotes e os compara às referências, o que transforma uma premissa da literatura em dado registrado. Retomando o plano acadêmico da justificativa (Seção 1), o artefato reúne as competências de engenharia de software do curso, do levantamento de requisitos e da modelagem de dados à sincronização, aos testes automatizados e à publicação em infraestrutura própria.

Do ponto de vista gerencial, cada indicador do aplicativo foi projetado para apoiar uma decisão concreta do pecuarista. O pasto que cumpriu o descanso indica para onde o lote pode seguir, e o lote há mais de 7 dias no mesmo piquete indica o momento de retirá-lo. A lotação da propriedade acima de 2,0 UA/ha sinaliza que a carga animal merece revisão, por venda ou por redistribuição dos lotes. A autonomia do insumo indica quando comprar a suplementação, antes que o estoque acabe, e o lote sem pasto aponta um registro a corrigir.

Esses indicadores respondem à falta de registro apontada na Introdução, em que o produtor sem método para registrar os custos perde o controle sobre os próprios ganhos (Lopes; Costa, 2024). Espera-se que o registro contínuo reduza essa perda, mas esse efeito não foi medido no uso real.

As limitações dividem-se em três grupos. Na avaliação, o aplicativo não foi avaliado em uso real, de modo que utilidade e usabilidade não foram medidas. Os requisitos vieram de um produtor e da vivência do autor, sem protocolo formal. Os cenários usaram um único modelo de celular, a API local e um segundo cliente simulado, e cada um foi repetido três vezes, o que não permite generalizar. Eles simulam um corte e um religamento da rede, e não oscilações repetidas, e a cobertura das telas é baixa. A busca bibliográfica não foi sistemática (Seção 3.3), e outros trabalhos correlatos podem ter ficado de fora.

No escopo, a entrega é só para Android e o login exige conexão. As regras de manejo são referências: o descanso parte do padrão da forrageira, sem considerar solo, região nem época do ano, e a ocupação e a lotação seguem os valores fixos de 7 dias e 2,0 UA/ha. Na estação seca, por exemplo, o aviso de rebrota cumprida pode indicar pasto pronto antes da hora. Na sincronização, a detecção de conflito atua sobre a edição de oito tipos de registro e compara o registro inteiro, e não campo a campo. Ficam de fora a criação e a exclusão, e as edições de aparelhos sem a versão de base seguem a regra da última escrita vence.

Como trabalhos futuros, o primeiro passo é a validação do aplicativo em uso real por produtores, com a revisão das regras de manejo por um zootecnista ou extensionista. Em seguida vêm repetições maiores dos cenários, incluindo a queda da rede durante o envio. Depois, estão a versão para iOS, o login sem conexão, o ajuste do descanso e da ocupação à época do ano e a detecção de conflito por campo, em vez de por registro, com verificação também da exclusão. A troca da detecção por versão por estruturas de dados replicadas sem conflito (*Conflict-free Replicated Data Types*, CRDT) é uma alternativa mais ampla, que dispensa a escolha do produtor.

6. Considerações Finais

Este trabalho perguntou como projetar e implementar um aplicativo móvel *offline-first* que integre lotes, pastagens rotacionadas e insumos e mantenha consistentes os dados sincronizados, diante da perda e do retorno da conexão. Como resposta, projetou-se e implementou-se o Pasto Livre, um aplicativo Android em Flutter que grava todo dado primeiro no aparelho e o envia a uma API REST por uma fila de saída quando há conexão. Nos testes executados, a arquitetura funcionou: os 663 testes do aplicativo e os 1.050 da API foram aprovados.

Entre os cinco critérios de avaliação, pesam mais a integridade dos dados sem conexão e a consistência da sincronização, pois são eles que sustentam a arquitetura proposta. A integridade foi mantida nos cenários A e B. A consistência, no sentido definido na Seção 2.5, ou seja, sem mutação perdida nem registro órfão, foi mantida para as mutações aceitas pelo servidor. Nos cenários em aparelho físico, as mutações criadas sem conexão chegaram ao servidor sem perda e na ordem causal declarada. A recusa de uma delas não deixou registro órfão no servidor, e o descarte do cadastro recusado desfez o efeito no aparelho. A edição concorrente do cenário D, que os critérios não cobrem e foi examinada à parte, também foi detectada: o servidor não sobrescreveu a edição mais recente, e a escolha ficou com o produtor. A detecção vale para o registro inteiro, e não por campo.

Os objetivos específicos foram atingidos nos limites da avaliação técnica. Os requisitos foram levantados com um produtor e com a vivência do autor, e as restrições de conectividade vieram da literatura (Seção 3.5.1). A estrutura de dados local, que sustenta a operação sem conexão, foi implementada em SQLite. O controle de rebanho por lotes, o histórico de manejo das pastagens, com ocupação e descanso dos piquetes, e o inventário de insumos com autonomia foram implementados, e os 17 requisitos funcionais foram atendidos nos testes (Seção 5.1). A sincronização com a API foi implementada e verificada pelos testes automatizados e pelos cenários de desconexão (Seções 5.3 e 5.2).

Diante da literatura, o trabalho sustenta que a operação local como regra é a escolha adequada para o público-alvo. Os correlatos de Chacon, Marinho e Alves (2023) e de Costa *et al.* (2024) tratam a conexão com o servidor como caminho principal, e os sistemas comerciais são de difícil uso para o produtor com pouca experiência com tecnologia (Costa *et al.*, 2024). Em propriedades onde o sinal falha, a inversão evita que o registro dependa da rede. O trabalho admite, porém, que a escolha tem custo: o conflito entre aparelhos passa a ser resolvido pelo produtor, e Freitas *et al.* (2024) avançaram mais na avaliação em uso real.

O aplicativo foi projetado para apoiar decisões concretas de manejo, como quando mover ou retirar um lote, rever a carga animal e comprar a suplementação (Seção 5.5), e seus recursos funcionam sem conexão, exceto no login. O trabalho não mediu esse efeito no uso real. O próximo passo é a validação com produtores, para medir a utilidade e a usabilidade que esta avaliação técnica não alcançou.

7. Referências

ANTHROPIC. Claude Code. 2026. Disponível em: https://www.anthropic.com/claude-code. Acesso em: 13 out. 2026.

ASSOCIAÇÃO BRASILEIRA DAS INDÚSTRIAS EXPORTADORAS DE CARNES (ABIEC). Beef Report 2026 revela novo patamar da pecuária brasileira. 2026. Disponível em: https://abiec.com.br/beef-report-2026-revela-novo-patamar-da-pecuaria-brasileira/. Acesso em: 22 out. 2026.

BLOC. Bloc concepts: Cubit. 2026. Disponível em: https://bloclibrary.dev/bloc-concepts/. Acesso em: 13 out. 2026.

BRAGA, G. J. *et al.* Métodos de pastejo e estimativas para o ajuste do número de bovinos na pastagem. Planaltina, DF: Embrapa Cerrados, 2020. (Circular Técnica, 46).

BROWN, S. The C4 model for visualising software architecture: component diagram. 2026. Disponível em: https://c4model.com/diagrams/component. Acesso em: 13 out. 2026.

BUAINAIN, A. M.; CAVALCANTE, P.; CONSOLINE, L. Estado atual da agricultura digital no Brasil: inclusão dos agricultores familiares e pequenos produtores rurais. Santiago: Comissão Econômica para a América Latina e o Caribe (CEPAL), 2021. (Documentos de Projetos, LC/TS.2021/61). Disponível em: https://hdl.handle.net/11362/46958. Acesso em: 2 out. 2026.

CENTRO DE ESTUDOS AVANÇADOS EM ECONOMIA APLICADA (CEPEA). PIB do agronegócio brasileiro apresenta crescimento de 12,20% em 2025. Piracicaba: CEPEA/ESALQ-USP, 2026. Disponível em: https://www.cepea.org.br/br/releases/pib-agro-cepea-pib-do-agronegocio-cresce-expressivos-12-em-2025.aspx. Acesso em: 28 abr. 2026.

CHACON, J.; MARINHO, É.; ALVES, I. AgroTec: um sistema de gestão pecuária para dispositivos móveis. In: CONGRESSO BRASILEIRO DE AGROINFORMÁTICA, 14., 2023, Natal. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2023. p. 334-341. DOI: 10.5753/sbiagro.2023.26576.

CONFEDERAÇÃO DA AGRICULTURA E PECUÁRIA DO BRASIL (CNA). Nota Técnica n. 27/2024: Plano Nacional de Identificação Individual de Bovinos e Búfalos – PNIB. Brasília, DF, 2024. Disponível em: https://www.cnabrasil.org.br/storage/arquivos/pdf/dtec.nt-27-plano-de-rastreabilidade-individual-de-bovinos.17dez2024.vf.pdf. Acesso em: 1 out. 2026.

COSTA, A. F. *et al.* Análise, modelagem e desenvolvimento de um sistema para gerenciamento de fazendas voltadas para a recria de gado. In: JORNADA CIENTÍFICA E TECNOLÓGICA DO IFSULDEMINAS, 16., 2024. Anais [...]. IFSULDEMINAS, 2024. ISSN 2319-0124. Disponível em: https://josif.ifsuldeminas.edu.br/ojs/index.php/anais/article/view/2419. Acesso em: 16 out. 2026.

COSTA, F. P. *et al.* Indicadores de desempenho na pecuária de corte: uma revisão no contexto da Plataforma +Precoce. Campo Grande, MS: Embrapa Gado de Corte, 2018. (Documentos, 237).

CRUZ, D. E. *et al.* Um estudo sobre a gestão dos insumos na criação de gado de corte em pequenas propriedades. Revista Gestão em Foco, Amparo, SP, n. 14, p. 256-266, 2022. Disponível em: https://portal.unisepe.com.br/unifia/wp-content/uploads/sites/10001/2022/03/20-UM-ESTUDO-SOBRE-A-GEST%C3%83O-DOS-INSUMOS-NA-CRIA%C3%87%C3%83O-DE-GADO-DE-CORTE-EM-PEQUENAS-PROPRIEDADES-256-a-266.pdf. Acesso em: 2 out. 2026.

DRIESSEN, V. A successful Git branching model. 2010. Disponível em: https://nvie.com/posts/a-successful-git-branching-model/. Acesso em: 1 out. 2026.

EUSÉBIO, G. dos S. *et al.* Análise dos determinantes da adoção de tecnologias digitais na pecuária intensiva de corte no Brasil. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE ECONOMIA, ADMINISTRAÇÃO E SOCIOLOGIA RURAL, 61., 2023, Piracicaba. Anais [...]. Brasília, DF: SOBER, 2023. DOI: 10.29327/sober2023.626971.

EXPRESS. Express: fast, unopinionated, minimalist web framework for Node.js. 2026. Disponível em: https://expressjs.com/. Acesso em: 2 out. 2026.

FERREIRA, V. B. S.; FERREIRA, C. A.; GRANDE, E. T. G. Estado da arte da pesquisa em: Clean Architecture e princípios de SOLID. Research, Society and Development, v. 11, n. 16, p. e335111637198, 2022. DOI: 10.33448/rsd-v11i16.37198.

FIELDING, R. T.; NOTTINGHAM, M.; RESCHKE, J. HTTP semantics. [S.l.], 2022. Disponível em: https://www.rfc-editor.org/rfc/rfc9110. Acesso em: 13 out. 2026.

FOWLER, M. Patterns of enterprise application architecture. Boston: Addison-Wesley, 2003.

FOWLER, M. Inversion of control containers and the dependency injection pattern. 2004. Disponível em: https://martinfowler.com/articles/injection.html. Acesso em: 13 out. 2026.

FREITAS, C. A. V. *et al.* O uso de sistemas de informações para auxílio em pastejo rotacionado. Revista Foco, v. 17, n. 11, e6746, p. 1-9, 2024. DOI: 10.54751/revistafoco.v17n11-023.

GONÇALVES, C. B. Q.; SCHLINDWEIN, M. M. Desenvolvimento de software para a tomada de decisão no setor agropecuário: uma revisão sistemática da literatura. Multitemas, Campo Grande, MS, v. 30, n. 75, p. 57-89, maio/ago. 2025. DOI: 10.20435/multi.v30i75.4735.

GOOGLE. Flutter: build apps for any screen. 2026. Disponível em: https://flutter.dev/. Acesso em: 2 out. 2026.

GRISE, M. M. *et al.* ABC corte program: sustainable intensification of beef production. Pesquisa Agropecuária Brasileira, v. 60, p. e02803, 2025. DOI: 10.1590/s1678-3921.pab2025.v60.04121. Disponível em: https://www.scielo.br/j/pab/a/M7gkJx66rKLk8RWQLcXpghj/. Acesso em: 2 out. 2026.

HEVNER, A. R. *et al.* Design science in information systems research. MIS Quarterly, v. 28, n. 1, p. 75-105, 2004. DOI: 10.2307/25148625.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA (IBGE). Censo Agro 2017: população ocupada nos estabelecimentos agropecuários cai 8,8%. Rio de Janeiro, 2019a. Disponível em: https://agenciadenoticias.ibge.gov.br/agencia-sala-de-imprensa/2013-agencia-de-noticias/releases/25789-censo-agro-2017-populacao-ocupada-nos-estabelecimentos-agropecuarios-cai-8-8. Acesso em: 1 out. 2026.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA (IBGE). Censo Agropecuário 2017: resultados definitivos, tabela 6778, número de estabelecimentos agropecuários, por tipologia. Rio de Janeiro, 2019b. Disponível em: https://sidra.ibge.gov.br/tabela/6778. Acesso em: 1 out. 2026.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA (IBGE). Pesquisa da Pecuária Municipal: tabela 3939, efetivo dos rebanhos, por tipo de rebanho. Rio de Janeiro, 2025. Disponível em: https://sidra.ibge.gov.br/tabela/3939. Acesso em: 1 out. 2026.

KLEPPMANN, M. *et al.* Local-first software: you own your data, in spite of the cloud. In: ACM SIGPLAN INTERNATIONAL SYMPOSIUM ON NEW IDEAS, NEW PARADIGMS, AND REFLECTIONS ON PROGRAMMING AND SOFTWARE (ONWARD!), 2019, Athens. Proceedings [...]. New York: ACM, 2019. p. 154-178. DOI: 10.1145/3359591.3359737.

LACERDA, D. P. *et al.* Design science research: método de pesquisa para a engenharia de produção. Gestão & Produção, São Carlos, v. 20, n. 4, p. 741-761, 2013. DOI: 10.1590/S0104-530X2013005000014.

LOPES, M. L. G.; COSTA, B. M. do N. Gerenciamento de custo em pequenas propriedades rurais: estudo voltado à atividade pecuária em regime familiar. Pensar Contábil, v. 26, n. 89, p. 35-44, 2024. Disponível em: http://www.atena.org.br/revista/ojs-2.2.3-08/index.php/pensarcontabil/article/view/4258. Acesso em: 30 out. 2026.

MARQUES, F. A. dos S. *et al.* Desenvolvimento de um aplicativo multilateral de gerenciamento de negócios para microempreendedores individuais do setor de estética. Research, Society and Development, v. 11, n. 16, p. e453111636898, 2022. DOI: 10.33448/rsd-v11i16.36898.

MARTHA JÚNIOR, G. B. *et al.* Área do piquete e taxa de lotação no pastejo rotacionado. Planaltina, DF: Embrapa Cerrados, 2003. (Comunicado Técnico, 101).

MARTIN, R. C. Clean architecture: a craftsman's guide to software structure and design. Boston: Prentice Hall, 2018.

MENDES, L. G. R.; MARTINS, A. D.; FREIRE, A. I. Manejo de pastagem rotacionado na pecuária de corte com ênfase no bem-estar do animal. Research, Society and Development, v. 11, n. 7, p. e42311730159, 2022. DOI: 10.33448/rsd-v11i7.30159.

RICHARDSON, C. Microservices patterns: with examples in Java. Shelter Island, NY: Manning, 2019.

SANTOS, P. M.; CORRÊA, L. de A.; BALSALOBRE, M. A. A. Guia prático para a implantação de sistemas de pastejo rotacionados para gado de corte. São Carlos, SP: Embrapa Pecuária Sudeste, 2005. (Documentos, 43).

SCHWABER, K.; SUTHERLAND, J. O Guia do Scrum: o guia definitivo para o Scrum: as regras do jogo. 2020. Disponível em: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-PortugueseBR-3.0.pdf. Acesso em: 2 out. 2026.

SILVA, L. F. dos S. Desenvolvimento de um aplicativo Android para gerenciamento de rebanho bovino de corte. 2022. Relatório final (Iniciação Científica em Tecnologia em Agronegócio) – Faculdade de Tecnologia de Botucatu, Botucatu, 2022.

VACANTI, D. S.; COLEMAN, J. The Kanban Guide. 2020. Disponível em: https://kanbanguides.org/the-kanban-guide/2020.12/pdf/kanban-guide.v2020.12.en.pdf. Acesso em: 2 out. 2026.