UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LUCAS BEDIN MARCHI

IMPLEMENTAÇÃO DE SINCRONISMO PARA UMA SOLUÇÃO MÓVEL DE GESTÃO PECUARISTA

MEDIANEIRA

2025

LUCAS BEDIN MARCHI

IMPLEMENTAÇÃO DE SINCRONISMO PARA UMA SOLUÇÃO MÓVEL DE GESTÃO PECUARISTA

Implementation of synchronization for a mobile livestock management solution

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Everton Coimbra de Araújo

MEDIANEIRA

2025

LUCAS BEDIN MARCHI

IMPLEMENTAÇÃO DE SINCRONISMO PARA UMA SOLUÇÃO MÓVEL DE GESTÃO PECUARISTA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 20/fevereiro/2025

Everton Coimbra de Araújo
Doutor
Universidade Tecnológica Federal do Paraná - Campus Medianeira

---

Cesar Angonese
Mestre
Universidade Tecnológica Federal do Paraná - Campus Medianeira

---

Ricardo Sobjak
Doutor
Universidade Tecnológica Federal do Paraná - Campus Medianeira

MEDIANEIRA

2025

Aos pequenos produtores, cujas mãos
calejadas tecem a história da nossa terra com
dedicação e honra, dedico este trabalho.

RESUMO

Este trabalho propõe uma arquitetura de sincronismo de dados *offline-first* para aplicações móveis em ambientes rurais, visando resolver desafios de conectividade intermitente na gestão pecuária familiar. A pesquisa justifica-se pela necessidade de garantir integridade e disponibilidade de dados em regiões com infraestrutura limitada de internet, onde soluções tradicionais dependentes de conexão contínua são inviáveis. O objetivo central foi desenvolver um mecanismo de sincronização bidirecional eficiente, utilizando SQLite para armazenamento local e PostgreSQL como banco remoto, mediado por uma *Application Programming Interface* (API) customizada para transferência e resolução de conflitos. A metodologia incluiu a modelagem da arquitetura, implementação da lógica de sincronização e testes em cenários simulados de desconexão. Os resultados demonstraram a eficácia do modelo na redução de latência (sincronização assíncrona), otimização de recursos de rede e manutenção da consistência dos dados, mesmo após longos períodos offline. Além disso, a solução eliminou dependências de serviços externos, aumentando a autonomia e escalabilidade do sistema. Conclui-se que a abordagem *offline-first*, aliada à sincronização customizada, é viável para democratizar ferramentas digitais em contextos rurais, assegurando resiliência e adaptabilidade a flutuações de conectividade. O estudo contribui com uma metodologia replicável para aplicações que demandem sincronismo robusto em ambientes desconectados, com potencial de auxiliar na modernização da agricultura familiar.

Palavras-chave: *offline-first*; sincronismo de dados; flutter; sqlite; agricultura familiar.

ABSTRACT

This work proposes an offline-first data synchronization architecture for mobile applications in rural environments, aiming to solve challenges of intermittent connectivity in family livestock management. The research is justified by the need to guarantee data integrity and availability in regions with limited internet infrastructure, where traditional solutions that depend on continuous connection are unfeasible. The main objective was to develop an efficient bidirectional synchronization mechanism, using SQLite for local storage and PostgreSQL as a remote database, mediated by a customized Application Programming Interface (API) for transfer and conflict resolution. The methodology included modeling the architecture, implementation of the synchronization logic and testing in simulated disconnection scenarios. The results demonstrated the effectiveness of the model in reducing latency (asynchronous synchronization), optimizing network resources and maintaining data consistency, even after long offline periods. In addition, the solution eliminated dependencies on external services, increasing the autonomy and scalability of the system. It is concluded that the offline-first approach, combined with customized synchronization, is viable for democratizing digital tools in rural contexts, ensuring resilience and adaptability to connectivity fluctuations. The study contributes with a replicable methodology for applications that require robust synchronization in disconnected environments, with the potential to assist in the modernization of family farming.

Keywords: offline-first; data synchronization; flutter; sqlite; family farming.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

Siglas

API	*Application Programming Interface*
CEPAL	Comissão Econômica para a América Latina e o Caribe
DTO	*Data Transfer Object*
HTTP	*Hypertext Transfer Protocol*
IU	Interface do Usuário
JPA	*Java Persistence API*
JSON	*JavaScript Object Notation*
MVC	*Model View Controller*
MVP	*Model View Presenter*
PWA	*Progressive Web Apps*
REST	*Representational State Transfer*
RFID	Identificação por Radiofrequência
RMAD	*Rapid Mobile App Development*
SDK	*Software Development Kit*
SOA	*Service-Oriented Architecture*
SQL	*Structured Query Language*
URL	*Uniform Resource Locator*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A agricultura familiar é definida por uma forma social específica de trabalho e produção. Esta forma é intrinsecamente ligada a um espaço geográfico definido, onde a interação entre um grupo familiar, unido por laços de parentesco, e a terra, juntamente com outros meios de produção, ocorre. Esta interação não se limita apenas à unidade familiar, mas se estende a outras unidades familiares e grupos sociais. Não é apenas uma atividade econômica, mas também a agricultura familiar é prática social que envolve a troca e a cooperação entre diferentes atores em uma comunidade (SCHNEIDER, 2016).

A agricultura de base familiar, que portanto além de uma forma de produção é modo de vida quanto no meio rural, tem ganhado cada vez mais reconhecimento. Sua importância e potencial para impulsionar um desenvolvimento rural dinâmico e voltado para a sustentabilidade estão sendo cada vez mais valorizados (SILVA; NUNES, 2023). No entanto, existem disparidades notáveis no ritmo e na extensão do progresso entre as diversas propriedades, sendo que aquelas que, mesmo pequenas, adotaram novas tecnologias progrediram mais do que as que não o fizeram.

A publicação na Comissão Econômica para a América Latina e o Caribe (CEPAL) de Buainain, Cavalcante e Consoline (2021) cita como as tecnologias digitais estão produzindo mudanças profundas na organização da sociedade, perpassando todas as esferas, da Economia à Cultura e argumenta que o setor agrícola, apesar de ser tradicionalmente considerado como retardatário em termos de adoção de inovações tecnológicas, não está imune a esse processo de transformação digital. Pelo contrário, as tecnologias digitais estão cada vez mais presentes no campo, alterando a maneira como a agricultura é praticada e gerenciada.

Tecnologias para a agropecuária podem simplificar o processo de criação animal e depender menos de mão de obra. Nesse contexto, a importância da inclusão tecnológica nos setores mais tradicionais é evidente. A adoção de tecnologias na gestão pecuarista tem o potencial de auxiliar na otimização da produção, melhorar a eficiência do uso de recursos e reduzir a dependência de mão de obra (DIAS; MALAFAIA; BISCOLA, 2020).

Lampert *et al.* (2015) discorre como é observável um aumento significativo na adoção de recursos computacionais como ferramentas de apoio à tomada de decisão em diversas áreas. Esse fenômeno se sobressai como uma segunda vertente promissora para a expansão da eficiência e produtividade no setor do agronegócio.

Considerando o amplo acesso à tecnologia móvel, mesmo em regiões remotas, a implementação de um aplicativo móvel torna-se uma estratégia viável. Este aplicativo, com uma interface prática, intuitiva e utilizando a arquitetura *offline-first* como base, que proporciona capacidade de sincronização de dados *offline* e *online*, pode proporcionar ao pequeno pecuarista uma visão abrangente do seu rebanho. Além disso, o aplicativo pode fornecer métricas importantes, que podem ser facilmente acessadas pelo usuário. Assim, a tecnologia móvel emerge como uma ferramenta valiosa para a gestão eficiente da pecuária.

1.1 Objetivo Geral e Específicos

O objetivo geral deste projeto é desenvolver a arquitetura *offline-first* como API e aplicar em uma aplicação para dispositivos Android de gestão de bovinos voltada para a pecuária de corte na agricultura familiar. Este objetivo pode ser dividido entre os seguintes objetivos específicos:

1. Desenvolver uma aplicação simplista de gestão pecuarista para servir como ambiente do sincronismo de dados;
2. Desenvolver e incrementar ao aplicativo um sistema de sincronização de dados robusto que permita operar tanto online quanto *offline*;
3. Analisar a competência da arquitetura *offline-first* e das tecnologias utilizadas para o cenário proposto.

1.2 Justificativa

A visão de muitos é que a revolução digital desencadeia muitas novas oportunidades para pequenos e médios produtores. No entanto, parece inquestionável o potencial de exclusão que este processo pode trazer, especialmente para os produtores que não conseguem se adaptar a estas tecnologias e se integrar às cadeias produtivas digitais que estão se tornando predominantes no agronegócio (BUAINAIN; CAVALCANTE; CONSOLINE, 2021).

A pecuária ainda é muito carente em termos de gestão. As anotações são precárias e não refletem o que está acontecendo no processo de produção, mascarando a situação (MALAFAIA, 2020). Portanto, uma aplicação móvel, que seja gratuita e de fácil manuseio, poderia trazer benefícios significativos, sobretudo para os pequenos produtores menos tecnificados da pecuária de corte na agricultura familiar. Esta ferramenta poderia ser um marco importante na modernização e eficiência de suas gestões.

1.3 Estrutura do trabalho

Este trabalho está estruturado em cinco capítulos, organizados da seguinte forma: o Capítulo 2 apresenta uma contextualização da pesquisa, abordando as tecnologias e metologias a serem utilizadas, com foco no desenvolvimento de software e a arquitetura *offile-first*. Posteriormente, será descrito um panorama da agricultura familiar e da bovinocultura de corte no Brasil, com ênfase no acesso e impacto da tecnologia. O Capítulo 3 detalha os materiais e métodos utilizados no desenvolvimento do projeto, descrevendo as etapas do processo de pesquisa e desenvolvimento. O Capítulo 4 apresenta os resultados do projeto, com foco no escopo do sistema, na apresentação do sistema e na implementação do sistema. Finalmente, o Capítulo

5 encerra o trabalho com a conclusão do projeto e as perspectivas futuras com os trabalhos futuros.

2 REFERENCIAL TEÓRICO

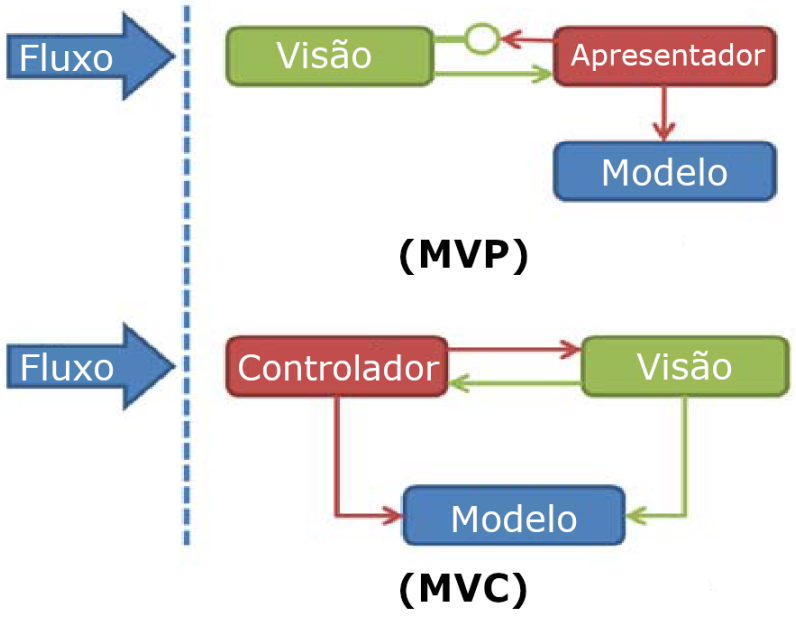
Neste capítulo, as técnicas de sincronização de dados e de desenvolvimento de software serão discutidas em detalhes, além das das tecnologias utilizadas. Também serão descritos os fundamentos da gestão na bovinocultura, com um foco particular na inclusão digital. Além disso, será feita uma revisão abrangente de trabalhos relacionados à área.

2.1 Padrões de Projeto

A escolha de padrões arquiteturais em aplicações web é crítica para equilibrar manutenibilidade e escalabilidade, especialmente em contextos com requisitos dinâmicos de interface. Neste projeto, adotou-se o *Model View Controller* (MVC) (Modelo-Visão-Controlador) em detrimento do *Model View Presenter* (MVP) (Modelo-Apresentador-Visão), justificada pela necessidade de desacoplamento entre lógica de negócios e interface, conforme discutido por Smith (2018).

No MVC, as requisições são direcionadas ao Controlador, que coordena Modelo (dados/regras) e Visão (interface), mitigando interdependências diretas. Essa separação é estratégica para cenários com atualizações frequentes de UI: alterações na camada de apresentação não exigem retrabalho na lógica de negócios, reduzindo riscos de regressão (SMITH, 2018). A Figura 1 ilustra essa dinâmica comparativamente ao MVP, onde o Apresentador centraliza a comunicação, mas exige recarregamento da página para atualizações ((QURESHI; SABIR, 2014)).

Figura 1 – Comparação MVC e MVP



Fonte: Adaptado de Qureshi e Sabir (2014).

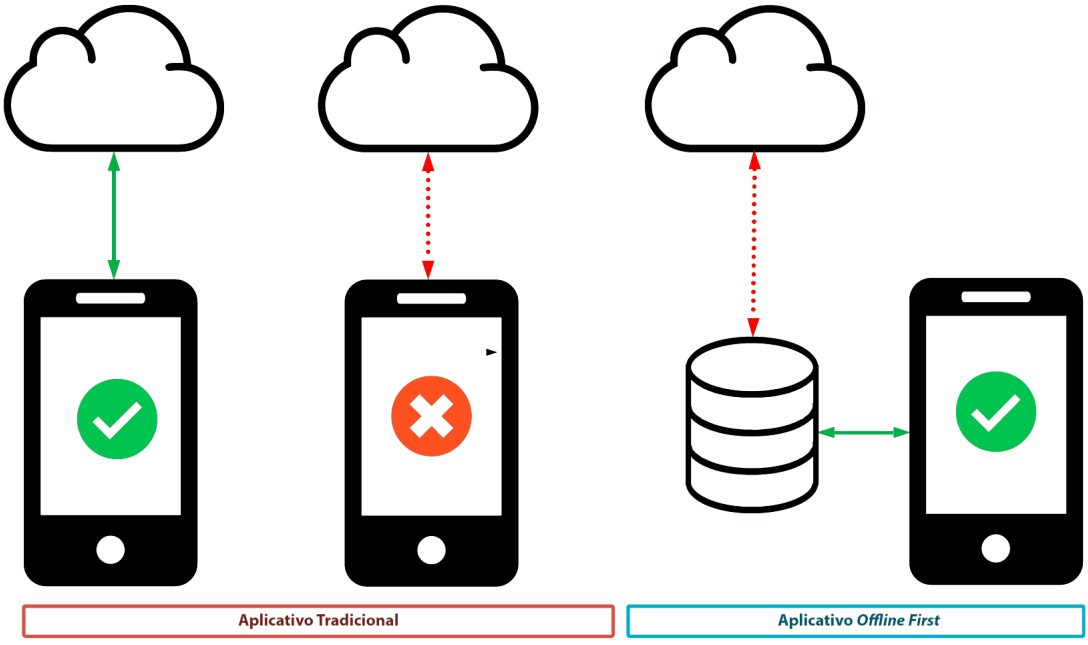
A opção pelo MVC alinha-se à necessidade de atualizações parciais da interface sem recarregamento total, requisito crítico em aplicações interativas como a desenvolvida. Enquanto no MVP a Visão desconhece o Modelo, limitando dinamismo, o MVC permite sincronização bidirecional ((QURESHI; SABIR, 2014)), essencial para funcionalidades como atualização constante de dados, funcionalidade central neste projeto.

2.2 Offline First

Cada vez mais, o acesso à internet através de dispositivos móveis é o mais crescente em detrimento dos *desktops*. Além disso, ao utilizar dispositivos móveis, é inevitável que haverá momentos em que se estará desconectado de uma rede de internet. No entanto, os desenvolvedores web costumam presumir que a conectividade de rede é garantida. O paradigma "*Offline-first*" inverte essa suposição (VANHALA *et al.*, 2017).

Segundo Vanhala *et al.* (2017), o termo "*Offline-first*" pode ser comparado a um aprimoramento progressivo, mas com ênfase na conectividade de rede como demonstrado na Figura 2. Primeiramente, é desenvolvida uma aplicação web que funciona completamente *offline*. Posteriormente, a experiência do usuário é aprimorada de forma gradual com recursos *online*. "*Offline-first*" é um paradigma de design relativamente novo para aplicações web.

Figura 2 – Aplicação Tradicional/Aplicação *Offline-first* Comparação



Fonte: Adaptado de Burgoyne (2023).

Lambert (2012) ressalta que a funcionalidade *offline* é um recurso que deve ser levada em consideração desde o início do processo de desenvolvimento. São apresentadas também três orientações para o desenvolvimento de aplicativos *offline* sob uma perspectiva técnica:

1. Transferir toda a lógica do servidor para o cliente. Utilizar o servidor apenas como um repositório de dados e estabeleça a comunicação entre o cliente e o servidor utilizando *JavaScript Object Notation* (JSON).

2. Desenvolver uma camada de abstração no lado do cliente para interagir com a API do lado do servidor.

3. Estabelecer uma camada de dados que coordene a solicitação de dados do servidor e o armazenamento em cache no navegador.

2.3 Aplicações Web Progressivas (PWA)

As *Progressive Web Apps* (PWA)s representam uma evolução nas práticas de desenvolvimento de aplicações web, combinando as melhores características dos aplicativos nativos com a flexibilidade e o amplo acesso das soluções web. Esse paradigma se baseia em princípios como progressividade, responsividade, segurança e, sobretudo, a capacidade de operar mesmo em condições de conectividade limitada, características essenciais para garantir uma experiência de usuário consistente.

Um dos pilares das PWAs é a utilização de *Service Workers*, scripts que funcionam como intermediários entre o navegador e o servidor, responsáveis por gerenciar o cache de forma inteligente. Essa abordagem permite que, após o acesso inicial, a aplicação mantenha conteúdos essenciais armazenados localmente, possibilitando o funcionamento offline ou em redes instáveis. Estratégias de cache, como as abordagens *cache-first* e *network-first*, são fundamentais para equilibrar a rapidez no carregamento e a atualização dos dados, oferecendo uma experiência similar à de aplicativos nativos.

Segundo Correia, Ribeiro e Silva (2021), no artigo "*Progressive Web Apps Development: Study of Caching Mechanisms*", evidencia que a implementação de técnicas avançadas de gerenciamento de cache pode proporcionar melhorias significativas na performance das aplicações. Por meio de métricas obtidas com ferramentas como o Google Lighthouse, que avalia parâmetros como o tempo de interação e a latência na resposta, o artigo demonstra que as PWAs apresentam desempenho superior em ambientes com conectividade comprometida quando comparadas a aplicações web tradicionais. Esses resultados reforçam a aplicabilidade dos conceitos de PWAs em cenários onde a confiabilidade e a continuidade do acesso aos dados são determinantes.

Adicionalmente, por não exigirem instalação via lojas digitais, as PWAs reduzem barreiras de acesso e facilitam a disseminação de soluções digitais, permitindo que o usuário acesse a aplicação diretamente pelo navegador. Essa característica é particularmente vantajosa em contextos onde a infraestrutura para aplicativos nativos é limitada ou onde há restrições quanto ao uso de dados e recursos computacionais. Essa abordagem pode ser especialmente relevantes em projetos que adotam a arquitetura *offline-first*, contribuindo para a criação de soluções

mais resilientes, eficientes e inclusivas, aptas a atender as demandas de ambientes com conectividade desafiadora.

2.4 Sincronismo de Dados

O sincronismo de dados é uma técnica utilizada para manter a consistência entre informações armazenadas em diferentes locais. Ela permite harmonizar dados entre uma fonte localizada em um lugar e um destino localizado em outro. Essa prática é aplicada em diversas situações, como a sincronização de dispositivos móveis e a sincronização de arquivos entre computadores pessoais. Em sistemas distribuídos, a sincronização de dados é fundamental para garantir que as alterações feitas em um banco de dados cliente/servidor ou vice-versa sejam devidamente atualizadas ao longo do tempo, mantendo a integridade dos dados (FAIZ; SHANKER, 2016).

Para Faiz e Shanker (2016) a sincronização geralmente é caracterizada pela prática de manter a integridade dos dados ou garantir que múltiplas cópias de um conjunto de dados estejam coerentes entre si, quando a se refere a área da ciência da computação. A sincronização de dados é frequentemente implementada por meio de primitivas de sincronização, como algoritmos que resolvem o desafio de dispositivos móveis que dependem de um único servidor. Quando o servidor não está disponível, esses dispositivos móveis podem operar no modo *offline*, utilizando os dados armazenados localmente. Assim que houver reconexão com o servidor, os dispositivos móveis podem facilmente sincronizar as alterações realizadas, como inserções, exclusões e atualizações, com os dados armazenados no servidor.

Quando se trata de manter dados em bancos de dados, tanto no dispositivo cliente quanto no servidor, ocorre alguma replicação das informações. Normalmente, após a troca de dados, o cliente é desconectado do servidor. No entanto, mesmo após essa desconexão, o banco de dados no dispositivo cliente pode ser modificado pelo Sistema Gerenciador de Banco de Dados (SGBD) implantado no próprio dispositivo. Isso ocorre porque, quando o cliente está conectado, qualquer alteração deve ser sincronizada com o servidor para garantir consistência. Da mesma forma, quando ocorrem alterações nos dados no servidor de banco de dados, elas também devem ser comunicadas ao dispositivo cliente. Isso é especialmente importante quando ambos estão trocando mensagens, para que o dispositivo possa receber uma resposta do servidor. O administrador de sincronização (como o *AnySync*) é um componente essencial do SGBD, pois garante a compatibilidade na replicação dos dados (SINGH; HASAN, 2019).

2.5 Java

A linguagem Java é uma escolha segura como tecnologia central deste e de muitos outros projetos, suas características atendem requisitos críticos de sistemas robustos: segurança

de tipos, portabilidade entre plataformas e gerenciamento automático de recursos. Esses atributos são particularmente relevantes para aplicações que demandam estabilidade em ambientes heterogêneos, conforme fundamentado por Gosling (2000).

Gosling (2000) discorre em seu livro como o Java é uma linguagem fortemente tipada e estaticamente tipada. Esta especificação faz uma distinção clara entre os erros detectáveis em tempo de compilação e aqueles que ocorrem em tempo de execução. O processo de compilação geralmente envolve a tradução de programas para uma representação de *byte code* independente de máquina. As atividades em tempo de execução incluem o carregamento e a vinculação das classes necessárias para executar um programa, a geração opcional de código de máquina, a otimização dinâmica do programa e a execução real do programa.

Como o Java é uma linguagem de alto nível, seus detalhes da representação da máquina não estão disponíveis através da linguagem. Ele inclui gerenciamento automático de armazenamento, geralmente usando um coletor de lixo, para evitar problemas de segurança da desalocação explícita. Implementações de coleta de lixo de alto desempenho podem ter pausas limitadas para suportar a programação de sistemas e aplicações em tempo real. A linguagem não inclui construções que não sejam seguras, como acessos a *array* sem verificação de índice, pois essas construções fariam um programa se comportar de maneira não especificada (GOSLING, 2000).

2.6 *JavaScript Object Notation* (JSON)

JSON é um formato de serialização de dados amplamente utilizado por programadores para codificar dados em diversos cenários, como na transferência de dados entre servidores e aplicações Ajax, e na comunicação entre servidores via serviços da Web. Em certo sentido, Brendan Eich inventou o JSON quando definiu a linguagem JavaScript e criou um formato serializado/literal combinado para valores, objetos e matrizes JavaScript (SEVERANCE, 2012).

Severance (2012) explica que na programação, as estruturas mais utilizadas, como variáveis escalares, listas lineares e pares de chave-valor, encontram em JSON uma representação natural e direta para serialização de dados. Essa característica minimiza a incompatibilidade entre as estruturas de dados na memória e o formato de serialização. Além de sua conveniência, JSON também se destaca por sua eficiência. Ao construir JSON em JavaScript, o formato ganha vantagem sobre outras alternativas, como *Extensible Markup Language* (XML), especialmente em aplicações que utilizam JavaScript.

2.7 Arquitetura Orientada a Serviços

MacKenzie *et al.* (2006) descreve como a *Service-Oriented Architecture* (SOA) é um paradigma que permite organizar e utilizar capacidades de sistemas distribuídos que podem

estar sob o controle de diferentes domínios de propriedade. De forma geral, entidades, como pessoas e organizações, desenvolvem capacidades para resolver ou apoiar soluções para os problemas que enfrentam em suas atividades comerciais. É comum que as necessidades de uma pessoa sejam atendidas pelas capacidades oferecidas por outra. No contexto da computação distribuída, os requisitos de um agente de computador são atendidos por um agente de computador de um proprietário diferente.

A falta de integração eficaz entre sistemas de informação pode levar a diversos problemas operacionais, tais como:

- Ineficiência Processual: A incapacidade de agilizar processos devido a limitações na integração de sistemas.
- Erros de Replicação: O risco de falhas ao replicar manualmente dados entre diferentes sistemas.
- Visão Fragmentada: A lentidão para detectar oportunidades e ameaças causada pela ausência de uma visão integrada dos sistemas.
- Complexidade de Atualização: A dificuldade em substituir infraestruturas de integração desatualizadas devido à sua complexidade.

Em resposta a esses desafios, Serman (2010) propõem a SOA como uma solução viável, promovendo a integração e a flexibilidade dos sistemas de informação, estabelecendo que uma arquitetura orientada a serviços é um princípio de arquitetura de software composta por componentes acessíveis por meio de interfaces genéricas e protocolos padronizados, preferencialmente sem restrições de licenças (os serviços). Esses componentes são projetados visando minimizar a dependência com os sistemas de informação que os utilizam e com a parte técnica do desenvolvimento, incentivando a reutilização e o aproveitamento de funcionalidades já existentes.

2.7.1 HyperText Transfer Protocol (HTTP)

O protocolo HTTP está por trás do WWW. Cada transação na web, seja uma solicitação de documento ou gráfico, um clique em um link de hipertexto ou o envio de um formulário, é mediada pelo HTTP. A Web é essencialmente uma plataforma para a distribuição de informações pela Internet, e o HTTP é o protocolo que facilita isso (WONG, 2000).

Segundo Wong (2000), uma das utilidades do HTTP é a sua capacidade de fornecer um meio padronizado para a comunicação entre computadores. A especificação do HTTP define como os clientes fazem solicitações de dados e como os servidores respondem a essas solicitações. Compreender o funcionamento do HTTP permite consultar servidores web manualmente e obter informações de baixo nível que os navegadores comuns ocultam dos usuários,

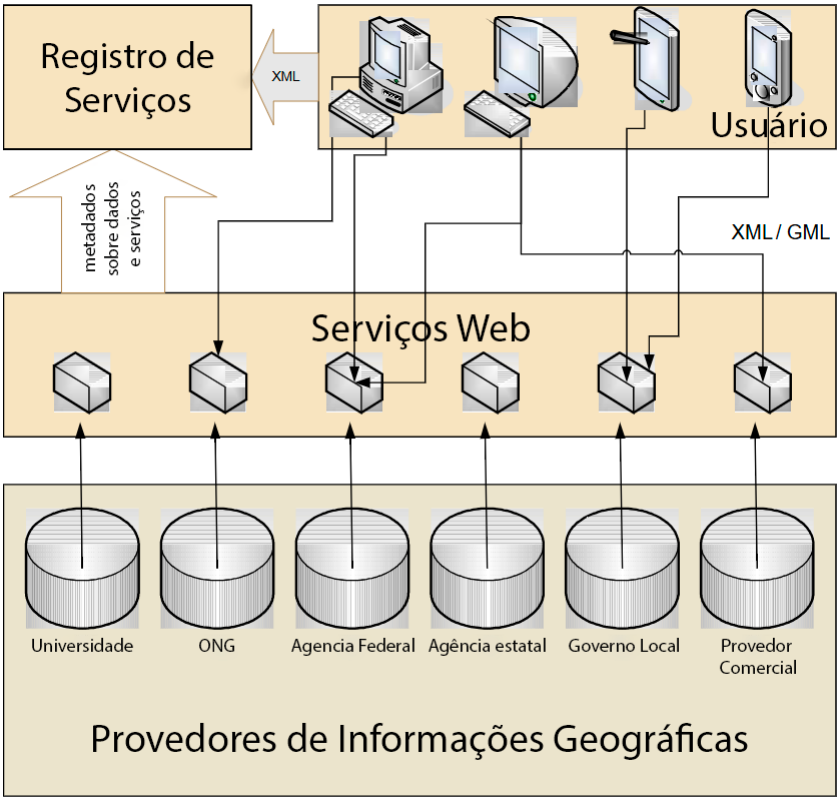
entender a interação entre clientes web (navegadores, robôs, motores de busca, etc.) e servidores web, além de simplificar os serviços da web para aproveitar ao máximo o protocolo.

2.7.2 Serviços Web

OLIVEIRA, OLIVEIRA e JUNIOR (2010) explica que um serviço Web é uma estrutura de software que pode ser identificada por um endereço universal, também conhecido como *Uniform Resource Identifier* (URI). Este serviço tem interfaces públicas e contratos que são estabelecidos e detalhados em XML. Essas definições podem ser encontradas por outros sistemas de software. Assim, esses sistemas podem interagir com o serviço Web de uma maneira determinada por sua definição, utilizando mensagens em XML que são transmitidas através de protocolos da Internet (apud WSA, 2003).

A Figura 3 demonstra o funcionamento a função dos serviços em uma IDE local, neste caso os possíveis provedores podem ser universidades, órgãos de governo, empresas comerciais, entre outros. Os serviços podem ser localizados através de um catálogo, onde estão registrados metadados de dados e serviços (OLIVEIRA; OLIVEIRA; JUNIOR, 2010).

Figura 3 – Serviços Web



Fonte: Adaptado de OLIVEIRA, OLIVEIRA e JUNIOR (2010).

Um modelo arquitetônico conhecido de Serviços Web é o *REST*, que é definido como um padrão para sistemas de hipermídia distribuídos, concebido inicialmente por Fielding (2000) em

sua tese. Segundo Garlan e Shaw (1993), um estilo arquitetônico é caracterizado pelo conjunto de componentes e conectores permitidos, bem como pelas restrições de sua combinação.

Fielding (2000) estabeleceu os seguintes atributos essenciais para o sucesso da *World Wide Web* (WWW)[1], que resultaram na formulação de restrições específicas, baseadas em estilos arquitetônicos preexistentes e uma adicional para a interface uniforme: Cliente-Servidor; Stateless; Arquitetura em Camadas; Cache; Código sob Demanda (opcional); Interface Uniforme. A Interface Uniforme, que é central para a API de serviços web *RESTful*, engloba quatro sub-restrições: Identificação de Recursos, Manipulação de Recursos por Representações, Mensagens Autodescritivas e Hipermídia. Um serviço web é considerado *RESTful* se adere a todas essas restrições, com a exceção do Código sob Demanda, que é facultativo (GIESSLER *et al.*, 2015).

2.7.3 Microsserviços

A expressão “Arquitetura de Microsserviços” cresceu nos últimos anos, caracterizando um método específico de criar aplicações de software como um conjunto de serviços que podem ser implantados de forma independente. Embora não exista uma definição exata para este estilo arquitetônico, existem algumas características comuns, como a organização baseada na capacidade de negócios, a implantação automatizada, a inteligência nos pontos finais e o controle descentralizado de linguagens e dados (LEWIS; FOWLER, 2014).

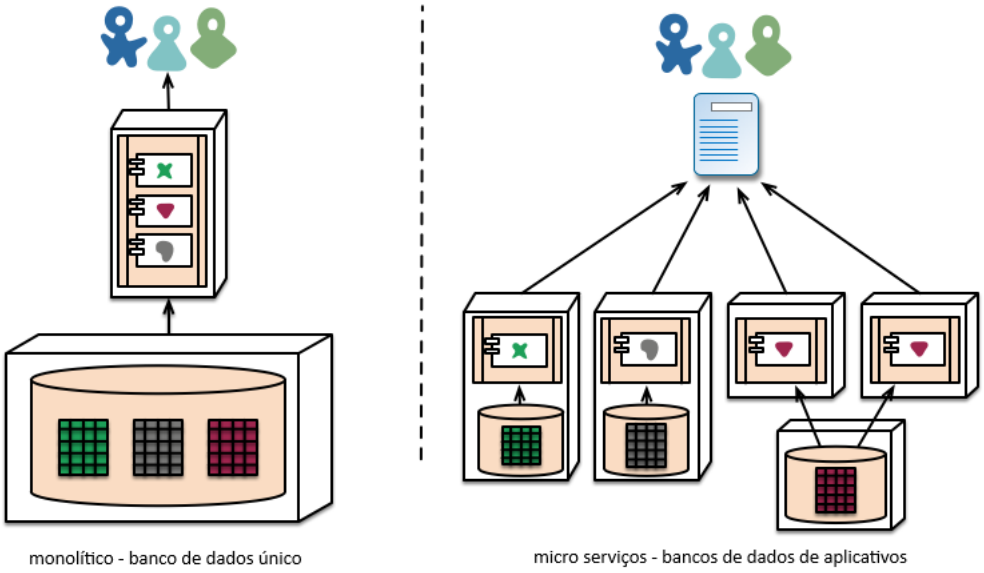
Segundo Lewis e Fowler (2014) a arquitetura de microsserviços é um estilo arquitetônico que facilita a criação de sistemas complexos através da decomposição em uma coleção de serviços menores. Cada microsserviço é encarregado de realizar suas próprias operações e se comunicar com os outros usando métodos de interação leves. A Figura 4 compara um sistema monolítico com um sistema de microsserviços.

Uma das características do micro serviços para Pereira, Back e Júnior (2018) é o gerenciamento descentralizado de dados, que, embora os aplicativos monolíticos geralmente optem por um único banco de dados lógico para gerenciar dados persistentes, os microsserviços adotam uma abordagem diferente. Eles preferem que cada serviço tenha a capacidade de gerenciar seu próprio banco de dados. Isso pode envolver o uso de instâncias diferentes da mesma tecnologia de banco de dados ou até mesmo sistemas de banco de dados completamente distintos. Essa abordagem é conhecida como Persistência Poliglota.

A Persistência Poliglota é uma estratégia que permite a um aplicativo usar diferentes tecnologias de armazenamento de dados que são mais adequadas às suas necessidades específicas. Isso pode proporcionar uma maior flexibilidade e eficiência, pois cada serviço pode escolher a tecnologia de banco de dados que melhor se adapta às suas necessidades e requisitos específicos. Embora a Persistência Poliglota possa ser implementada em um aplicativo monolítico, ela é mais comumente associada aos microsserviços. Isso ocorre porque os mi-

[1] Em português: Rede Mundial de Computadores.

Figura 4 – Monolítico/Microsserviços Comparação



Fonte: Adaptado de Lewis e Fowler (2014).

crosserviços, por sua natureza, são projetados para serem independentes e autônomos, o que se alinha bem com a ideia de permitir que cada serviço gerencie seu próprio banco de dados (PEREIRA; BACK; JÚNIOR, 2018).

2.7.4 API Gateway

O *API*[2] *Gateway* é um serviço robusto de gerenciamento e controle, centralizado e orientado por API, que atua como um ponto de controle na fronteira do sistema. Antes do surgimento e popularização do conceito de microsserviços, o *API Gateway* já existia. Atualmente, seu principal cenário de aplicação é o OpenAPI, uma plataforma aberta para parceiros externos (ZHAO; JING; JIANG, 2018).

Para Zhao, Jing e Jiang (2018), quando o conceito de microsserviços se popularizou, o *API Gateway* emergiu como um componente essencial para a integração na camada de aplicação superior. O uso do serviço *API Gateway* em microsserviços traz inúmeros benefícios. Ele permite que o cliente permaneça imune à localização da instância do serviço e oculta a maneira como a aplicação é fragmentada em vários microsserviços. Ele oferece a API mais adequada para cada cliente, minimizando as solicitações. Uma das vantagens é a capacidade de encapsular a estrutura interna da aplicação. Em comparação com a chamada de um serviço específico, a interação do cliente com o gateway é mais simplificada. O *API Gateway* disponibiliza uma API específica para cada cliente, diminuindo a quantidade de comunicações entre cliente e servidor e simplificando o código do cliente.

[2] Em português: Interface de Programação de Aplicação.

2.7.5 API Web Storage

Para compreender a *API Web Storage*, é útil revisitar seu predecessor, o curiosamente denominado *cookie*. Os *cookies* de navegador, nomeados em referência a uma técnica de programação antiga para transferir pequenos valores de dados entre programas, o "*cookie mágico*", são um meio integrado de transmitir valores de texto de um servidor para um cliente. Os servidores podem utilizar os valores armazenados nesses *cookies* para rastrear informações do usuário em páginas da web. Os valores dos *cookies* são enviados sempre que um usuário visita um domínio. Por exemplo, os *cookies* podem guardar um identificador de sessão que permite a um servidor web identificar a qual usuário pertence um determinado carrinho de compras, armazenando um ID único em um *cookie* do navegador que corresponde ao banco de dados do carrinho de compras do servidor. Assim, à medida que o usuário navega de uma página para outra, o carrinho de compras pode ser atualizado de maneira consistente. Outra aplicação para os *cookies* é armazenar valores locais em um aplicativo, de modo que esses valores possam ser utilizados em carregamentos subsequentes da página (LUBBERS *et al.*, 2010).

Para Lubbers *et al.* (2010), a *API Web Storage* se destaca pois utiliza uma API simplificada em que os desenvolvedores têm a capacidade de armazenar valores em objetos *JavaScript* que são facilmente recuperáveis e que persistem durante o carregamento da página. Seja utilizando *sessionStorage* ou *localStorage*, os desenvolvedores podem escolher se esses valores persistirão durante o carregamento da página em uma única janela ou aba, ou entre reinicializações do navegador, respectivamente. Os dados armazenados não são transmitidos pela rede e podem ser facilmente acessados em visitas subsequentes a uma página. Adicionalmente, valores maiores podem ser mantidos usando a API de armazenamento na Web, com capacidade de até alguns *megabytes*. Tornando o armazenamento na Web apropriado para dados de documentos e arquivos que excederiam rapidamente o limite de tamanho de um *cookie*.

2.8 Flutter Framework

Flutter é um *framework* multiplataforma do Google para desenvolvimento ágil de aplicativos móveis com desempenho nativo. Sua principal utilidade está na capacidade de criar interfaces uniformes para Android e iOS a partir de um único código-base, reduzindo custos e tempo de desenvolvimento. Diferente de outras soluções, o Flutter não depende de componentes nativos ou tecnologias web, utilizando seu próprio motor de renderização para garantir consistência visual e alta performance em diferentes dispositivos (TASHILDAR *et al.*, 2020).

A estrutura é centrada em *widgets*, elementos modulares que definem a interface e a lógica de interação. *Widgets stateless* (imutáveis) e *stateful* (dinâmicos) permitem a construção de interfaces flexíveis: enquanto os primeiros são ideais para elementos estáticos, os segundos habilitam atualizações em tempo real, como formulários ou animações, sem reiniciar a aplicação (BOUKHARY; COLMENARES, 2019). Para otimizar o desempenho em interfaces complexas, o

Flutter emprega técnicas como a renderização seletiva, evitando recarregar toda a árvore de componentes quando apenas um elemento é alterado.

O *Stateful Hot Reload* é um recurso-chave para produtividade, permitindo que desenvolvedores visualizem alterações no código instantaneamente, sem perder o estado atual do aplicativo. Isso acelera a depuração e a experimentação de interfaces (SHARMA *et al.*, 2022). Para integração com serviços externos, o pacote Dio simplifica requisições HTTP, oferecendo interceptadores, gerenciamento de erros e suporte a operações assíncronas de forma intuitiva (AMEEN; MOHAMMED, 2022).

Em projetos de grande escala, o padrão *Business Logic Component* (Bloc) organiza o fluxo de dados entre a interface e fontes externas (APIs, bancos de dados), separando claramente a lógica de negócio da camada visual. Isso facilita a manutenção e a escalabilidade, evitando a propagação desnecessária de estados entre componentes (AMEEN; MOHAMMED, 2022). Combinado à capacidade de compilação direta para código nativo, o Flutter posiciona-se como uma escolha consciente para o desenvolvimento de aplicativos robustos.

2.9 Agricultura Familiar

A expressão ‘agricultor familiar’ é mais do que apenas um conceito, é uma construção política. Ela é idealizada pelas políticas públicas como uma forma de representação, mas muitas vezes está distante da realidade social desses indivíduos, que também são conhecidos por outras denominações, como ‘camponeses’ ou ‘pequenos agricultores’ (NEVES, 2007).

O assunto da agricultura familiar é um campo manancial para estudos, é possível identificar diversas contribuições que discorrem o entendimento sobre sua diversidade interna. Savoldi e Cunha (2010) apresentam três principais tipificações para o conceito de agricultura familiar:

- Família Agrícola de Caráter Empresarial: Essa categoria refere-se ao conhecido como o “verdadeiro agricultor”. Essas famílias possuem uma estrutura econômica, social, técnica e patrimonial que lhes permite investir em produção rentável voltada principalmente para o mercado.
- Família Camponesa: O objetivo principal dessas famílias é a manutenção da produção agropecuária e da propriedade familiar. Elas não seguem os padrões produtivistas de mercado, mas sim orientam suas práticas com base em valores tradicionais.
- Família Agrícola Urbana: Esse grupo tem seus próprios valores que guiam a produção, focando na qualidade de vida. Eles não ignoram a realidade do mercado, mas também não abandonam os valores da família camponesa.

Dentro das modalidades de produção familiar, a pecuária se destaca por seu papel significativo no sustento e continuidade das famílias rurais globalmente, pois é uma fonte vital de receita e soberania alimentar, envolvendo a criação de gado (GARTZIA *et al.*, 2016). No Brasil,

essa prática produtiva está presente em 78,9% (4.006.656) das propriedades rurais, sendo que 77,1% (3.089.452) pertencem à agricultura familiar, conforme os dados do Censo Agropecuário de 2017 (IBGE, 2018).

2.10 Bovinocultura

A bovinocultura, que se refere à criação e manejo de bovinos, abrange tanto a produção de carne quanto a produção de leite. Dentro do setor pecuário voltado à criação de bovinos, existem dois tipos de sistemas de produção, sendo elas a de corte e a leiteira:

- Bovinocultura de Leite: A criação de gado leiteiro segundo Miglior, Loker e Shanks (2013) envolve a escolha criteriosa e o acasalamento de animais que atendam aos objetivos de criação estabelecidos, visando modificar as características genéticas das próximas gerações e elevar a eficiência econômica do processo. Um exemplo de meta de criação é o desenvolvimento de linhagens que, além de aumentarem a produção leiteira, apresentem melhorias significativas em saúde e capacidade reprodutiva. A seleção é direcionada para aqueles exemplares cuja prole possa garantir um retorno financeiro superior, resultado de uma produção mais eficiente e custos reduzidos, atribuídos à robustez e à fertilidade otimizadas dos animais.

- Bovinocultura de Corte: No Anuário CiCarne da cadeia produtiva da carne bovina de 2023, MALAFAIA e BISCOLA (2023) explica como a criação de bovinos para produção de carne no Brasil começou em seu próprio descobrimento, na chegada dos animais trazidos pelos colonizadores portugueses. Logo no começo, a sua evolução ocorreu sustentada por decisões tomadas pelos próprios criadores. Entretanto, engajados na busca por qualidade, indústrias, associações e instituições de ciência e tecnologia atuaram de maneira decisiva para evoluir do setor. Hoje, o Brasil passou a ser autossuficiente na produção de carne bovina. Cerca de 72,1% da produção é destinada ao mercado interno, que possui uma das mais elevadas médias de consumo de carne por ano, chegando a 36,7 kg por habitante ao ano.

Essas atividades fazem parte significativa da agropecuária brasileira, com 27,3% das propriedades agropecuárias no país dedicadas exclusivamente à criação de bovinos para corte. Ela desempenha um papel crucial na economia rural, fornecendo emprego e renda para milhares de famílias. Além disso, a produção de carne e leite contribui para a segurança alimentar e nutricional da população (MALAFAIA, 2020).

O manejo de bovinos sempre foi mais casual por motivos de tradição, fundamentada na experiência e sabedoria dos próprios agricultores, obtida através da experiência de vida e passada de geração em geração através da prática comunitária, do trabalho colaborativo e das tradições. Uma consequência disso é que a grande maioria dos agricultores brasileiros, inde-

pendentemente do tamanho de suas operações, atua como pessoa física e não como entidades jurídicas formalmente estabelecidas (BUAINAIN; CAVALCANTE; CONSOLINE, 2021).

Outro grande motivador desse cenário é o acesso à informação e a disponibilidade de infraestrutura de internet. Apenas 12,1% dos estabelecimentos possuíam acesso à internet, enquanto 31,2% contavam apenas com rádio e 52,5% dispunham de televisão. No que diz respeito à agricultura familiar e às regiões Norte e Nordeste, esses indicadores mostram um cenário ainda mais desafiador e restritivo para inovações: 9%, 31,4% e 51,6%, respectivamente, para acesso à internet, rádio e TV entre os agricultores familiares. Entre os agricultores das regiões Norte e Nordeste, os números são de 6,4% e 5,1% com internet; 25,7% e 25,3% com rádio; e 41,2% e 45,6% com televisão (BUAINAIN; CAVALCANTE; CONSOLINE, 2021).

2.11 Trabalhos Relacionados

Camargo, Silva e Caetano (2015) conduziram o desenvolvimento do aplicativo para dispositivos Android, IOS e Windows Phone, o BovGen, um protótipo de aplicativo destinado a auxiliar pequenos pecuaristas no gerenciamento de gado utilizando o método de pastejo rotacionado. O BovGen oferece funcionalidades como controle e visão geral do rebanho, dados do peso do animal em arroba, troca de piquetes e análise de melhores preços com base em gráficos. A solução tecnológica desenvolvida demonstrou potencial em elevar a eficiência dos pecuaristas, permitindo-lhes ter uma visão ampla sobre o gado de corte e o método de pastejo rotacionado através das informações obtidas pelo aplicativo como o de gerenciamento completo de gado, gerenciamento do sistema rotativo, informações importantes de preços, facilitação do processo rotineiro, facilitação dos cálculos, padronização em tarefas, permitindo-lhes implantar processos adequados ao pequeno produto.

O software desenvolvido por Boeira e Cantarelli (2016) visa aprimorar o controle e manejo dos dados produzidos no setor de gestão de pecuária. Utilizando a metodologia *Feature Driven Development*, a aplicação foi construída com PHP e MySQL, incorporando brincos de identificação animal com chip de Identificação por Radiofrequência (RFID) para facilitar o gerenciamento individualizado e coleta eletrônica de precisão. O resultado é uma ferramenta que promove uma gestão mais eficiente para o pecuarista, simplificando atividades e permitindo um controle mais preciso sobre os animais e a propriedade, contribuindo para uma pecuária mais rentável e sustentável. O software desenvolvido demonstrou ser intuitivo e abrangente, oferecendo desde o gerenciamento financeiro até o controle de estoque e manejo de bovinos, evidenciando a importância da tecnologia na modernização da gestão pecuária.

O artigo de Marques e Ferreira (2017) aborda o uso de tecnologias da informação para a gestão rural, o aplicativo ISApp foi criado para manter os conceitos da planilha eletrônica ISA, com o objetivo de facilitar a coleta de dados em campo. É acessível em nuvem e foi desenvolvido para ser intuitivo e fácil de usar, mesmo por usuários iniciantes. O ISApp foi desenvolvido em HTML5 numa plataforma *Rapid Mobile App Development* (RMAD), permitindo uso em dis-

positivos móveis como iOS e Android. É possível a inserção de dados off-line, geolocalização automática dos resultados, e apresentação dos dados em mapas e relatórios além de poder ser integrado com outros sistemas como o Google Maps e MAPinr, facilitando o mapeamento e a navegação em áreas rurais. Os resultados com base em gráficos de indicadores socioeconômicos indicam que o ISApp facilita a coleta de dados e a tomada de decisões, contribuindo para a melhoria da assistência técnica e do gerenciamento dos agroecossistemas.

Carvalho (2019) discorreu sobre o desenvolvimento de um sistema *offline* para gestão da pecuária de corte e fluxo de caixa para pequenas propriedades rurais. O projeto visa fornecer uma ferramenta tecnológica que auxilie pequenos produtores rurais na gestão eficiente de suas atividades pecuárias sem a necessidade de conexão com a internet. O sistema proposto é dividido em módulos que permitem o gerenciamento detalhado do rebanho, incluindo controle individual dos animais, manejo sanitário, reprodução, pesagem, e transações financeiras. A solução foi desenvolvida utilizando Java e Spring Boot, com foco em usabilidade e acessibilidade para os usuários finais. O resultado é um sistema robusto que pode ser operado localmente, promovendo a autossuficiência dos produtores em suas tomadas de decisão gerenciais.

Correia *et al.* (2020) apresentaram o desenvolvimento de um aplicativo móvel destinado a auxiliar produtores rurais na gestão da produtividade pecuária através de indicadores zootécnicos. O artigo destaca a importância do agronegócio para o PIB brasileiro e a necessidade de melhorar a produtividade na pecuária, que ainda é considerada deficiente. Utilizando a metodologia PWA e o Ionic Framework, os autores desenvolveram um protótipo que permite aos usuários inserir dados do rebanho e receber informações sobre desempenho e produtividade. O aplicativo visa ser multiplataforma e facilitar o processo de tomada de decisão em propriedades rurais, com ou sem acesso à conexão móvel. Os resultados preliminares indicam que a ferramenta está em fase de desenvolvimento e promete ser um recurso valioso para o setor agropecuário.

Nandyal *et al.* (2021) propuseram uma arquitetura de sincronização móvel em nuvem para gestão de objetos de grande porte (100MB a 1GB) utilizando o armazenamento de objetos OpenStack Swift. O estudo destaca a implementação de um serviço de sincronização (*Mobile Sync*) que emprega técnicas de fragmentação (*chunking*), segmentação e compressão para otimizar o armazenamento e transferência de dados em ambientes com conectividade intermitente. Desenvolvido em Java com integração a bancos de dados locais (SQLite e LevelDB), o *framework* permite operações *offline* e sincronização assíncrona, garantindo consistência eventual por meio de registros de versões e metadados. Os resultados experimentais demonstraram redução de 63,2% no tempo de upload e 92,9% no download em comparação a soluções como Parse Server e BaasBox, além de suportar arquivos de até 5GB via segmentação no Swift. A abordagem elimina dependências de hardware especializado e oferece APIs para manipulação parcial de objetos, sendo relevante para aplicações rurais que demandam transferência robusta de dados volumosos, como imagens de monitoramento ou registros zootécnicos em alta resolução.

Prado *et al.* (2021) desenvolveram o aplicativo e-SUS Vacinação, uma solução móvel voltada para a otimização de registros vacinais durante a pandemia de COVID-19 no Brasil. Utilizando o *framework* Flutter, a aplicação adotou uma arquitetura *offline-first*, permitindo o registro de dados sem conexão à internet e sincronização assíncrona com o sistema nacional de saúde quando a conectividade é restaurada. O desenvolvimento seguiu metodologia ágil SCRUM, integrando funcionalidades como leitura de documentos via câmera com reconhecimento de imagem baseado em *machine learning* (operacional offline), autopreenchimento de campos repetitivos (dados do profissional e lote da vacina) e validação automática de inconsistências. A inversão do modelo de dados tradicional, priorizando o cadastro em massa de cidadãos imunizados com o mesmo imunobiológico reduziu de 19 para 4 campos manuais por ficha, alcançando 84% de automação. A arquitetura foi reforçada pelo armazenamento local em SQLite e sincronização via APIs REST, garantindo resiliência em áreas remotas. O uso do Flutter facilitou a portabilidade para Android e iOS, enquanto a integração com o *backend* do e-SUS APS PEC assegurou compatibilidade com a Rede Nacional de Dados em Saúde (RNDS). Resultados preliminares indicaram redução de erros em registros, economia de tempo para profissionais de saúde e qualificação de políticas públicas baseadas em dados confiáveis, alinhando-se à Lei Geral de Proteção de Dados (LGPD) pelo tratamento local de informações sensíveis.

O trabalho de Santos (2021) objetiva o desenvolvimento de um software para gestão de dados, previsão e gerenciamento na bovinocultura de corte, visando auxiliar no processo de tomada de decisões pertinentes à atividade. O programa, nomeado como SISGEBO, foi desenvolvido para os sistemas operacionais Windows Vista, Windows 7 e Windows 8, utilizando a arquitetura MVC e as tecnologias Bootstrap, Angular, Electron e Basel. Os dados de referência foram coletados a partir de modelos de variáveis produtivas e econômicas obtidas em literaturas específicas e de planilhas desenvolvidas no contexto da bovinocultura de corte. O resultado é um sistema de interface intuitiva em modelo Web para desktops sem fins lucrativos.

Brotherton *et al.* (2022) desenvolveram o Hikma Health EHR, um sistema de registros eletrônicos de saúde *offline-first* e de código aberto para populações deslocadas, utilizando React Native, TypeScript e Python integrados a bancos de dados SQLite (local) e PostgreSQL (nuvem). O sistema emprega uma arquitetura de sincronização bidirecional baseada em timestamps para resolver conflitos, permitindo atualizações locais e transferência assíncrona via APIs REST quando a conectividade é restaurada. Implementado em clínicas móveis no Líbano e Nicarágua, o Hikma Health suportou 26.000 pacientes, oferecendo fluxos de trabalho modulares para diferentes especialidades médicas e interfaces multilíngue (inglês, árabe, espanhol). Resultados destacaram melhoria de 40% na eficiência de atendimento e redução de 65% em erros de documentação comparado a sistemas manuais, além de recursos como busca difusa (*fuzzy search*) para nomes com variações ortográficas. A abordagem *offline-first* com compressão de dados e armazenamento hierárquico em GCS (Google Cloud Storage) mostrou-se particular-

mente relevante para contextos rurais com infraestrutura limitada, oferecendo *insights* aplicáveis à gestão de dados pecuários em áreas remotas.

O sistema de Miranda (2022) foi projetado com base em entrevistas, análise de documentos, livros e dados, com o objetivo de torná-los sistematizados para auxiliar pecuaristas no manejo bovino. O software desenvolvido permite o registro persistente de informações, facilitando o acesso via dispositivos conectados à internet. Utilizando engenharia de software, o sistema foi implementado com Java, PostgreSQL, e tecnologias web como HTML e JavaScript. Os resultados destacam uma interface intuitiva e a capacidade do sistema de realizar operações de dados essenciais, como cadastro, consulta, atualização e remoção, contribuindo para uma gestão eficiente na pecuária de corte.

Chacon, Marinho e Alves (2023) demonstram o desenvolvimento e a aplicação do aplicativo AgroTec, um sistema inovador de gestão pecuária projetado para dispositivos móveis Android e IOS utilizando tecnologias já consolidadas como HTML, CSS e JavaScript e o *framework* Ionic, visando otimizar a rotina dos produtores rurais. Através de uma interface intuitiva e recursos avançados, o sistema permite o monitoramento eficiente do rebanho, controle de produção e gestão financeira. Os resultados obtidos demonstram uma melhoria significativa na produtividade e na tomada de decisões, com redução de custos operacionais e aumento da lucratividade. Este trabalho evidencia o potencial do AgroTec em transformar a gestão agropecuária, alinhando tecnologia e sustentabilidade.

3 MATERIAIS E MÉTODOS

Neste capítulo, são apresentados os recursos e procedimentos utilizados para a realização deste estudo, as fases do plano, e as principais arquiteturas e tecnologias utilizadas.

3.1 Materiais

Nesta seção, são descritos os materiais a serem empregados na realização deste projeto.

3.1.1 Penpot

Ao desenvolver uma interface para um aplicativo, inicialmente são feitos *wireframes*, que são os rascunhos iniciais da interface prevista para a aplicação, após isso é importante o desenvolvimento de um protótipo de maior nível de fidelidade. O Penpot[1] é uma plataforma de código gratuita para esse propósito, ele oferece elaboração de design de interfaces e prototipagem de alto nível. O Penpot tem capacidade de criar componentes reutilizáveis, isso significa que um elemento, como um botão, pode ser criado e replicado em diversas partes do design. Diferentes estilos e comportamentos podem ser definidos para quando o usuário interage com o componente, adicionando elementos de *feedback* visual e tornando a experiência do usuário mais dinâmica. Adicionalmente, permite a criação de estilos globais, como cores e tipografias, que podem ser aplicados em todo o projeto, garantindo consistência e agilidade nas alterações, além de permitir a criação de estados interativos para os componentes, como *hovers* e cliques.

É possível criar protótipos interativos, simulando a interação do usuário com o design. Links, transições de tela e animações podem ser adicionados para simular uma experiência mais realista do projeto em desenvolvimento, e ainda possibilita exportar ativos diretamente do Penpot.

Além disso, o Penpot permite que os desenvolvedores inspecionem o design, obtendo informações precisas sobre tamanhos, cores e estilos, facilitando a transformação do design em código. Devido a essas características, o Penpot foi selecionado como a ferramenta de prototipagem para a elaboração do conceito inicial do aplicativo proposto neste trabalho.

3.1.2 Visual Studio Code

Um editor de código-fonte leve, que funciona em *desktop* e está disponível para Windows, macOS e Linux. O Visual Studio Code[2] (versão 1.89) vem com suporte integrado para

[1] Site oficial do Penpot: https://penpot.app/

[2] Site oficial do Visual Studio Code: https://code.visualstudio.com/

JavaScript, TypeScript e Node.js e possui um rico ecossistema de extensões para outras linguagens e tempos de execução. Como o Dart, por exemplo.

O Visual Studio Code oferece conclusões inteligentes para variáveis, métodos e módulos importados. Sua depuração é interativa, permitindo que percorra o código-fonte, inspecione variáveis, veja pilhas de chamadas e execute comandos no console.

Ele permite habilitar idiomas adicionais, temas, depuradores, comandos e muito mais através de extensões para personalizar cada recurso de acordo com suas necessidades. Além de ter suporte para Git, permitindo o trabalho com controle de versão sem sair do editor.

3.1.3 Dart

Dart[3] é uma linguagem de programação orientada a objetos e baseada em classes desenvolvida e mantida pelo Google, focada em desenvolvimento *front-end* para web e mobile. É a linguagem por trás do *framework* Flutter, que permite a criação de aplicativos nativos compilados para várias plataformas a partir de uma única base de código. Tem uma sintaxe que pode ser familiar para desenvolvedores de outras linguagens como Java ou C#. Entre suas vantagens, está a capacidade de compilar para JavaScript ou para código nativo, o que contribui para a performance e versatilidade dos aplicativos desenvolvidos.

Dart é formado por elementos como variáveis, operadores, comandos condicionais, *loops*, funções, classes, objetos e enumerações. Ele incorpora conceitos cruciais de uma linguagem de programação orientada a objetos, como herança e programação genérica. O *Software development kit* (SDK)[4] Flutter do próprio Google e a conhecida ferramenta de publicidade Google Ads são programados com Dart. Exemplos adicionais incluem os sites do New York Times e do Groupon.

Dart pode funcionar em todos os navegadores móveis e *desktop* modernos. Para programar com esta linguagem, tudo que é necessário é de um editor de texto simples. Sua sintaxe semelhante à de uma linguagem humana o torna mais fácil de ler. Ele usa menos comandos, mas mais opções. A livre escolha de nomes para variáveis torna mais compreensível qualquer código escrito. Isso também elimina a necessidade de fazer muitos comentários no código. Ele admite usar espaços, guias e quebras de linha como desejar. Isso permite que um programador estruture claramente o código de uma forma que será ignorada pelo compilador. Salvo á exceções, como palavras-chave, nomes de variáveis e nomes de funções (ou seja, termos definidos no Dart como '*if*', '*string*' e '*void*', entre outros).

[3] Site oficial do Dart: https://dart.dev/

[4] Em português: kit de desenvolvimento de software

3.1.4 Flutter

Flutter[5] é uma plataforma de desenvolvimento de aplicativos móveis. Foi lançado publicamente em 2016 pelo Google, que escolheu o Flutter como sua estrutura de nível de aplicativo para o seu mais novo sistema operacional, o Fuschia. Os aplicativos Flutter podem ser executados também em iOS e Android.

O Flutter suporta o uso de pacotes compartilhados contribuídos por vários desenvolvedores para os ecossistemas Flutter e Dart. Isso permite construir rapidamente o aplicativo sem criar tudo do zero. Os pacotes existentes permitem muitas categorias de uso, por exemplo, fazer solicitações de rede HTTP, lidar com roteamento/rota personalizado com o Fluro, uma biblioteca de roteamento Flutter segura para nulos, integrar-se com APIs de programação de dispositivos Android, como lançador de *Uniform Resource Locator* (URL)[6] universal e bateria, e usar *Software Development Kit* (SDK)s de plataforma de terceiros.

O Flutter utiliza Dart, que possui vários recursos úteis, como *mixins*, e tem seus próprios componentes de interface do usuário, juntamente com um motor para renderizá-los nas plataformas iOS e Android.

3.1.5 Android Studio

Android Studio[7] é considerado a ferramenta oficial de desenvolvimento de aplicativos Android. É um ambiente de desenvolvimento disponível para Windows, macOS e GNU/Linux. O Android Studio disponibiliza diversas funções, como um editor de layout avançado: WYSIWYG, capacidade de trabalhar com componentes de Interface do Usuário (IU) usando arrastar e soltar, função de visualização de layout em várias configurações de tela. Ele possui uma estrutura de aplicação baseada em Gradle, diferentes tipos de estruturas e várias gerações de arquivos *Android Application Pack* APK, um analisador de código estático nomeado de Lint, que permite encontrar problemas de desempenho, incompatibilidades de versão e muito mais.

Com modelos de código e integração com GitHub, o Android Studio ajuda a criar recursos comuns de apps e importar exemplos de código, além de trazer diversas possibilidades com *frameworks* e ferramentas de teste.

3.1.6 SQLite

Uma biblioteca em processo que implementa um mecanismo de banco de dados SQL autossuficiente, sem servidor, sem configuração e transacional. O código do SQLite[8] está no

[5] Site oficial do Flutter: https://flutter.dev/
[6] Em português: Localizador Uniforme de Recursos
[7] Site oficial do Android Studio: https://developer.android.com/
[8] Site oficial do SQLite: https://www.sqlite.org/

domínio público e é, portanto, gratuito para uso para qualquer finalidade. Não requer uma configuração separada e é transacional, suporta transações de Atomicidade, Consistência, Isolamento, Durabilidade (ACID), o que significa que todas as transações são atômicas (ou todas as alterações em uma única transação ocorrem, ou nenhuma ocorre) e sobrevivem a falhas do sistema e do aplicativo.

O SQLite é muito leve, tornando-o fácil de usar como software embarcado com dispositivos como televisões, telefones celulares, câmeras, dispositivos eletrônicos domésticos, etc. As operações de leitura e gravação do SQLite são quase 35% mais rápidas do que o sistema de arquivos.

Sua portabilidade alcança todos os sistemas operacionais de 32 bits e 64 bits e arquiteturas *big-* e *little-endian*. O SQLite é muito fácil de aprender. Essas vantagens tornaram o SQLite a escolha de banco de dados local na aplicação proposta, enquanto para a sincronização *online*, será utilizado o MySQL.

3.1.7 PostgreSQL

PostgreSQL[9] é um sistema de gerenciamento de banco de dados relacional-objeto (ORDBMS) de código aberto, lançado inicialmente em 1996. Reconhecido por sua conformidade rigorosa com padrões *Structured Query Language* (SQL) e suporte a recursos avançados, como tipos de dados personalizados, consultas complexas e transações ACID, ele se destaca em cenários que exigem alta confiabilidade e flexibilidade. Sua arquitetura escalável permite ajustes de desempenho por meio de técnicas como particionamento de tabelas, replicação e paralelização de consultas, adaptando-se a cargas de trabalho variáveis.

Por ser de código aberto e distribuído sob a licença PostgreSQL License[10], o software possui uma comunidade global ativa, responsável por contribuições contínuas, desde extensões especializadas (como PostGIS para geolocalização) até otimizações de segurança. Essa abertura facilita a integração com diversas linguagens de programação (Python, Java, C/C++) e *frameworks*, tornando-o versátil para aplicações empresariais, científicas e web.

O PostgreSQL é amplamente utilizado em ambientes que demandam operações complexas, como sistemas geográficos, análise de *big data* e soluções financeiras. Sua capacidade de lidar com grandes volumes de dados e garantir consistência em transações concorrentes (via *Multi-Version Concurrency Control* (MVCC)) o torna uma escolha comum em projetos críticos.

[9] Site oficial do PostgreSQL: https://www.postgresql.org/

[10] Licença *Berkeley Software Distribution* (BSD) modificada, que permite uso e modificação sem restrições de código proprietário.

3.1.8 Docker

Docker[11] é uma plataforma de código aberto para desenvolvimento, implantação e execução de aplicações em ambientes isolados chamados *containers*. Seu uso no projeto foi fundamental para garantir consistência entre ambientes de desenvolvimento, teste e produção, eliminando discrepâncias causadas por diferenças em sistemas operacionais ou dependências de software.

A tecnologia de containerização permite empacotar aplicações com todas as bibliotecas, *frameworks* e configurações necessárias em imagens portáteis. No contexto do GadoApp, Docker foi empregado para isolamento de serviços, especificamente do banco de dados PostgreSQL.

3.1.9 Spring Boot

O Spring Boot[12] é um projeto Spring que visa simplificar o processo de configuração e publicação de aplicações baseadas no ecossistema Spring. Ele foi escolhido por ser muito utilizado e simplificado no processo de criação de serviços Web RESTful. Além disso, ele possui diversas utilidades, como fornecer recursos prontos para produção, como métricas, verificações de integridade e configuração externalizada, que são bem úteis na etapa de desenvolvimento.

O Spring Boot fornece dependências iniciais opinativas para simplificar a configuração de compilação, e configura automaticamente as bibliotecas Spring e *3rd party* sempre que possível. Sua biblioteca Spring é amplamente usada para a criação de microsserviços Web. Ele fornece um conjunto de ferramentas e extensões autorais e de terceiros para facilitar o desenvolvimento de aplicativos.

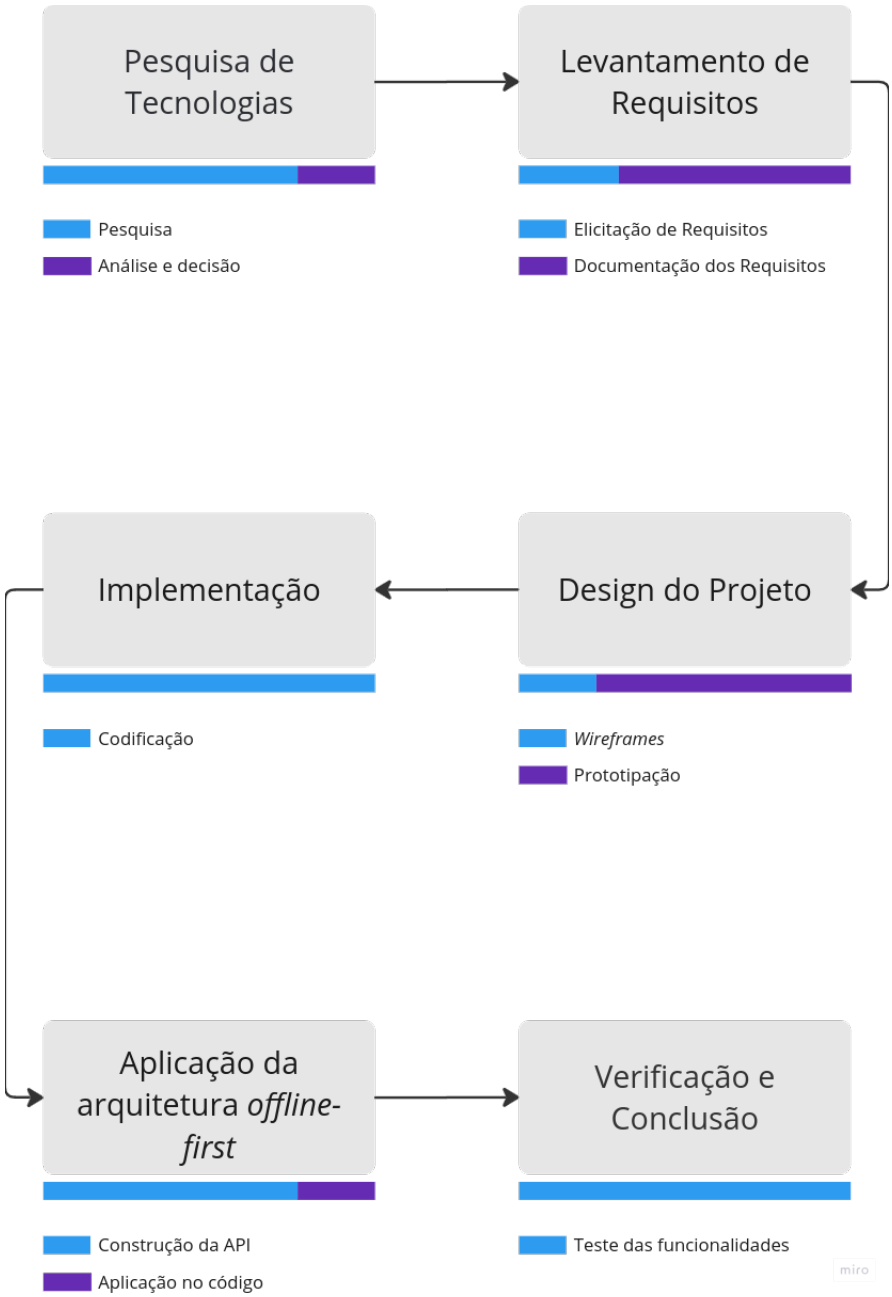
3.1.10 Insomnia

Insomnia[13] é uma ferramenta de código aberto para desenvolvimento, teste e documentação de APIs. Sua escolha no projeto deve-se à capacidade de simular e validar solicitações HTTP complexas, gerenciar variáveis de ambiente, recursos essenciais para testar a API de sincronismo desenvolvida. Além disso, oferece integração nativa com versionamento via Git e suporte a testes unitários diretamente na interface, facilitando a colaboração e a manutenção da consistência durante a implementação. A ferramenta foi preferida em detrimento de alternativas por seu foco em customização e leveza, aliado à ausência de custos para funcionalidades avançadas.

[11] Site oficial do Docker: https://www.docker.com/
[12] Site oficial do Spring Boot: https://spring.io/guides/gs/spring-boot
[13] Site oficial do Insomnia: https://insomnia.rest/

3.2 Metodologia

Figura 5 – Diagrama de Fluxo



Fonte: Autoria própria (2025).

O desenvolvimento do projeto seguiu o processo ilustrado no Diagrama de Fluxo (Figura 5). Inicialmente foi realizada uma pesquisa para selecionar as tecnologias adequadas que viabilizaram o estudo proposto. Esta seleção foi fundamentada em uma análise comparativa das opções disponíveis, levando em consideração fatores como funcionalidades, familiaridade e compatibilidade com os objetivos do projeto. Os detalhes desta seleção, bem como as tecnologias escolhidas, estão descritos na seção 3.1.

Posteriormente à fase inicial de investigação, procedeu-se a definição dos requisitos, tanto funcionais quanto não funcionais, que deveriam nortear o desenvolvimento do projeto. Essa etapa foi fundamentada em uma análise detalhada das demandas específicas do público-alvo, bem como um estudo comparativo de trabalhos correlatos.

A execução do processo de design do projeto sucedeu na plataforma Penpot, começando com a criação de *wireframes* que serviram como um esboço preliminar. Esses *wireframes* progressivamente refinados, incorporando elementos de design modernos e interfaces intuitivas, foram evoluídos para o protótipo do projeto. A normas de design científico foram aplicadas para atender as necessidades funcionais e também promover uma interação fluida, essencial para a aceitação no contexto inserido.

A fase de implementação do projeto se iniciou logo após a definição do design. O desenvolvimento do *front-end* foi realizado utilizando o *framework* Flutter e linguagem Dart, utilizando a IDE Visual Studio Code como ambiente único nessa etapa de desenvolvimento. Na implementação do aplicativo, a interface foi elaborada para fornecer ao usuário um *dashboard* onde ficaram as principais informações e sobre o rebanho, como: quantidade de bovinos, status e quantidade de rebanhos. Foi implementado também cadastro com informações relevantes do animal, edição e remoção de rebanhos e bovinos em rebanhos.

A arquitetura *offline-first* foi implementada utilizando o SQLite para armazenamento local e a sincronização de dados foi desenvolvida através da criação de uma API REST que recebe os dados do armazenamento local e envia para o banco de dados do lado do servidor e armazenado com PostgreSQL. A sincronização dos dados foi gerenciada pelo Spring Boot, para maior segurança e integridade dos dados durante a sincronização. Após a aplicação da arquitetura *offline-first*, foram realizados testes para analisar sua funcionalidade e sua capacidade de se adaptar ao contexto proposto. Os testes visaram validar a persistência local dos dados, a sincronização automática com o servidor e a integridade das informações. Os resultados dos testes foram descritos no Capítulo 4.

Após o desenvolvimento da arquitetura proposta, iniciou-se os testes da API utilizando a ferramenta Insomnia, para analisar sua funcionalidade e a sua capacidade de se adaptar no contexto imposto, tentado realizar operações através do protocolo HTTP. Os testes da API visaram validar os dados de retorno; validar os *headers* da resposta; validar se a resposta está de acordo; validar se quando o *content-type* é alterado, o comportamento continua o mesmo; validar se a estrutura do JSON está correta; validar se quando der erro o status está de acordo com os códigos de erro; validar se uma requisição com informações incompleta, qual será o comportamento da requisição, afim de saber quais comportamentos serão esperados da API.

4 RESULTADOS

Este capítulo apresenta os resultados do estudo, com foco central na avaliação do mecanismo de sincronismo de dados *offline-first* desenvolvido, analisando sua eficiência, confiabilidade e adaptabilidade a cenários de conectividade intermitente. Inicialmente, será detalhado o escopo do sistema, evidenciando as funcionalidades implementadas e o potencial de contribuição dessa ferramenta para a área de gestão pecuarista. Em seguida, será mostrada a modelagem inicial do sistema, que serviram como base para o desenvolvimento da aplicação do sincronismo. Por fim, a apresentação do sistema demonstra o resultado obtido com o desenvolvimento da aplicação do sincronismo de dados e testes da API implementada.

4.1 Escopo do Sistema

O aplicativo nomeado "GadoApp"foi pensada para facilitar a gestão de rebanhos de corte, oferecendo um conjunto de funcionalidades básicas, incluindo cadastro, edição e exclusão de bovinos e rebanhos. Uma característica distintiva do GadoApp é a sua capacidade de operar de forma autônoma, sem a necessidade de conexão constante à internet. Os dados inseridos no aplicativo são armazenados localmente no dispositivo Android e sincronizados automaticamente com um servidor central quando uma conexão de rede estiver disponível, garantindo a atualização e a segurança das informações.

De início, o gerenciador vai trabalhar apenas com bovinocultura de corte, focando apenas nas métricas mais importantes para esse tipo de cultura. O sistema deve incluir valores como peso, idade, raça, genitores e um identificador para cada animal registrado, assim como para qual rebanho ele será atribuído, rebanho esse que já deve estar previamente cadastrado no sistema.

A interface do GadoApp foi desenvolvida com foco na usabilidade e na facilidade de navegação, tornando-o acessível a usuários com diferentes níveis de familiaridade com tecnologias digitais. A aplicação oferece uma experiência intuitiva e eficiente, permitindo que os pecuaristas acessem e gerenciem as informações de seus rebanhos de forma rápida e prática.

4.2 Modelagem do Sistema

A definição dos requisitos é essencial para delimitar o escopo de um projeto de desenvolvimento, guiando as etapas de planejamento, implementação e validação do software. No caso do aplicativo GadoApp, a especificação dos requisitos funcionais e não funcionais priorizou a praticidade de uso, com foco nas funcionalidades essenciais para a primeira versão do aplicativo. Buscou-se, assim, atender às necessidades básicas dos usuários, permitindo o gerenciamento eficiente de seus rebanhos. O Quadro 1 descreve uma das funcionalidades prin-

cipais do aplicativo, o cadastro de bovinos, com seus requisitos funcionais e não funcionais detalhados.

Quadro 1 – Requisito funcional 1

Requisito Funcional
RF01 Controlar animal: Organizar todos os dados do animal.
Requisitos Não Funcionais
RNF1.1 O sistema deverá ter um CRUD para animal.
RNF1.2 Os campos a serem preenchidos para cada cadastro são: identificador, status, sexo, raça (opcional), peso (opcional), data de nascimento, pai (opcional), mãe (opcional), observação (opcional) e o rebanho em que o animal pertence.
RNF1.3 O identificador pode ser o brinco do animal ou apenas uma string.

Fonte: Autoria própria (2025).

O Quadro 2 detalha os requisitos para o sistema de cadastro de rebanhos no aplicativo. Os requisitos não funcionais presentes descrevem o comportamento esperado do sistema nesta funcionalidade, incluindo informações sobre os campos obrigatórios para o cadastro e as demais funcionalidades que permitem o gerenciamento completo dos rebanhos.

Quadro 2 – Requisito funcional 2

Requisito Funcional
RF02 Controlar rebanho: Organizar todos os dados de um rebanho.
Requisitos Não Funcionais
RNF2.1 O sistema deverá ter um CRUD para rebanho, com um campo de nome.
RNF2.2 Dentro da área de rebanho o usuário deverá poder adicionar os animais que quiser nesse rebanho, utilizando uma consulta e seleção.

Fonte: Autoria própria (2025).

Por fim, o Quadro 3 apresenta os requisitos para a implementação da arquitetura *offline-first*, essencial para o funcionamento do aplicativo. Essa arquitetura garante a persistência dos

dados, tanto localmente quanto em nuvem, permitindo o acesso e a manipulação das informações mesmo sem conexão com a internet, e sincronizando-as automaticamente quando a conexão é reestabelecida.

Quadro 3 – Requisito funcional 3

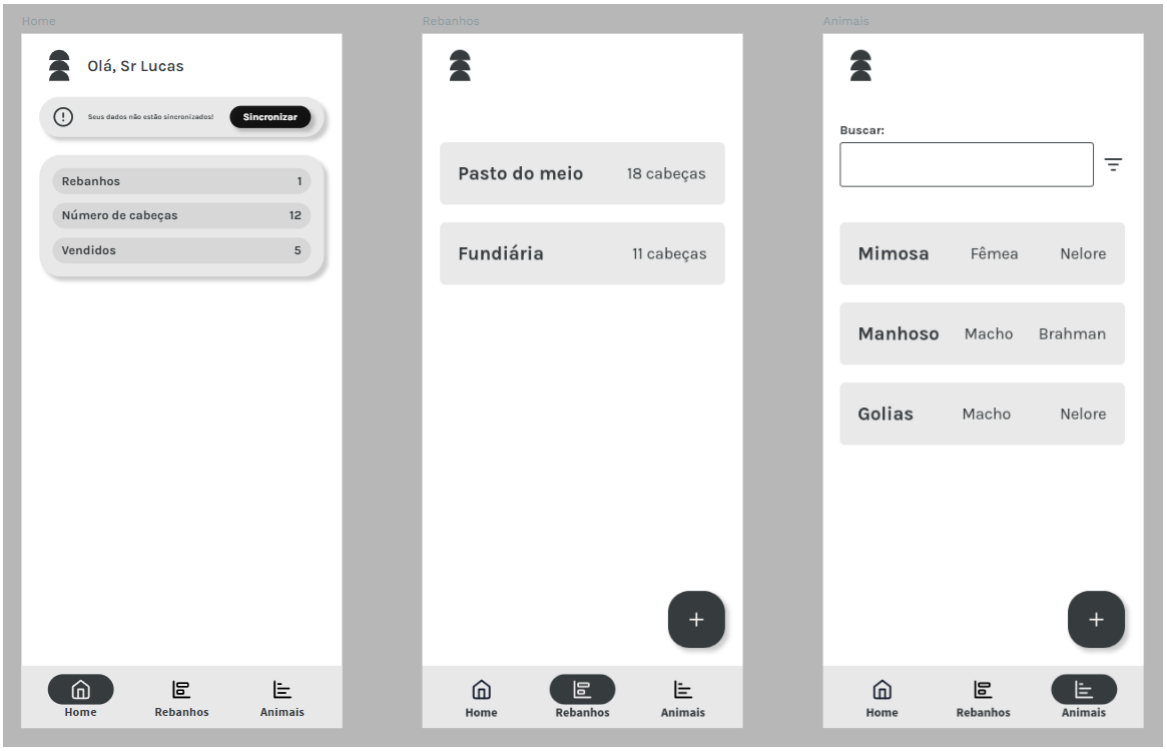
Requisito Funcional
RF03 Sincronizar dados localmente e em nuvem: Acesso total ao sistema mesmo sem conexão.
Requisitos Não Funcionais
RNF3.1 O sistema deve possibilitar a edição, remoção e adição dos dados em todo o sistema mesmo sem acesso à internet.
RNF3.2 O sistema deve sincronizar os dados local com os dados em núvem quando a conexão for reestabelecida.
RNF3.3 O sistema deve lidar com conflitos de dados.

Fonte: Autoria própria (2025).

Após a definição dos requisitos, o desenvolvimento do sistema avançou para a etapa de prototipação (Figura 6). Nessa fase, o foco principal residiu em garantir a usabilidade e a melhor experiência possível para o usuário. Através de um processo iterativo de design, alguns protótipos foram elaborados, explorando diferentes combinações de cores, layouts e elementos visuais. A escolha final do design considerou um esquema de cores de alto contraste, que facilita a leitura e a identificação dos elementos na tela, e uma organização visual intuitiva, seguindo as convenções de design mais atuais e amplamente adotadas em aplicativos *mobile*. Essa abordagem visa minimizar a curva de aprendizado do usuário, tornando a interação com o sistema mais natural, independentemente do seu nível de familiaridade com tecnologias.

Para atuar como banco de dados do lado do servidor, adotou-se o PostgreSQL, implantado em um contêiner Docker configurado em ambiente de desenvolvimento simulado. Essa arquitetura garantiu isolamento de recursos, portabilidade e consistência entre diferentes estágios do projeto. No esquema relacional, foram modeladas duas entidades principais: Rebanhos e Bovinos. A relação entre as tabelas foi estabelecida por meio de chaves estrangeiras, assegurando a integridade referencial — um bovino pertence a um rebanho específico, enquanto um rebanho pode conter múltiplos bovinos. A Figura 7 ilustra detalhadamente esse diagrama entidade-relacionamento (DER), incluindo cardinalidades, tipos de dados e vínculos hierárquicos.

Figura 6 – Protótipo Inicial

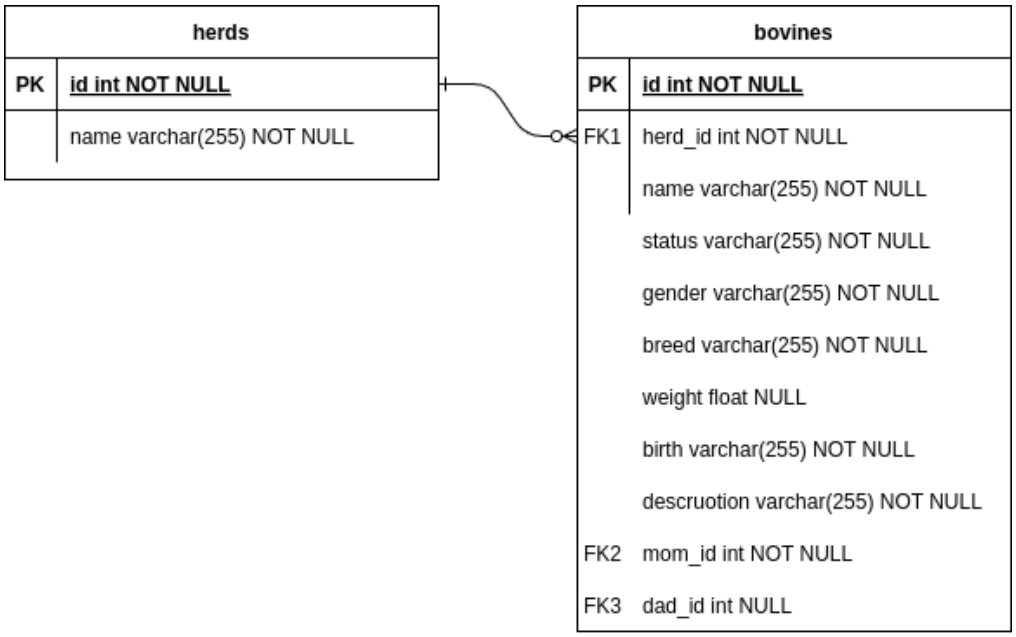


Fonte: Autoria própria (2025).

4.3 Apresentação do Sistema

A interação inicial com o sistema se dá por meio de uma tela de autenticação e login que foi desenvolvida, utilizando o *framework* Flutter, empregando seus componentes nativos para

Figura 7 – Diagrama entidader-elacionamento (DER) do banco de dados PostgreSQL



Fonte: Autoria própria (2025).

assegurar uma experiência de usuário intuitiva e familiar. A simplicidade da interface visa facilitar o processo de autenticação, solicitando ao usuário apenas o endereço de e-mail e a senha correspondente. Para garantir a integridade e a validade dos dados inseridos, foram implementadas validações de entrada. O campo de e-mail exige a conformidade com um formato válido, seguindo um padrão de máscara específico, enquanto o campo de senha impõe um requisito de comprimento mínimo de seis dígitos, visando reforçar a segurança das contas. A Figura 8 ilustra a interface da tela de login e a próxima tela que é apresentado ao usuário, a "Início", evidenciando os campos de entrada e o layout geral da tela.

Após a autenticação bem-sucedida, o usuário é redirecionado para a tela inicial da aplicação, que apresenta um painel de controle (*dashboard*) conciso com informações relevantes sobre os rebanhos e animais cadastrados. As informações exibidas incluem o número total de rebanhos cadastrados, a quantidade total de cabeças de gado presentes nesses rebanhos e o número de animais vendidos. A estrutura da tela é baseada no componente `BottomNavigationBarItem` do Flutter, um padrão de navegação amplamente adotado em aplicativos móveis contemporâneos, caracterizado por uma barra inferior contendo botões que permitem a transição entre as diferentes seções da aplicação. Essa escolha de design visa proporcionar uma navegação intuitiva e consistente, familiarizando o usuário com a interface. A tela inicial também exibe um título que apresenta o nome da aplicação, além de um botão posicionado à direita que possibilita o encerramento da sessão (sair da conta) e o retorno à tela de login. A disposição desses elementos oferece ao usuário um ponto de acesso rápido para as principais informações e para a gestão dos dados.

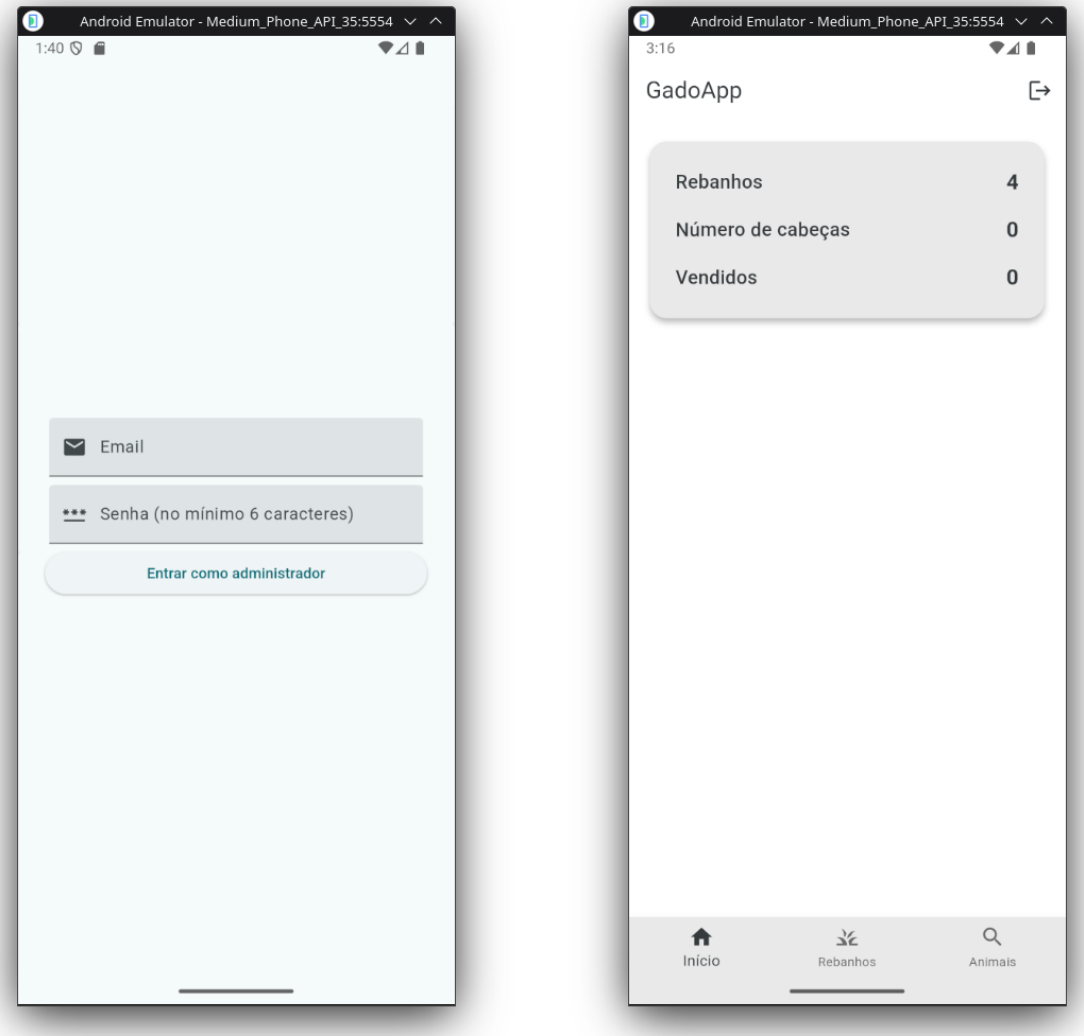
A próxima seção acessível da aplicação demonstrada na Figura 9 é a tela de listagem de rebanhos, que exibe todos os rebanhos previamente cadastrados pelo usuário, juntamente com a quantidade de animais vinculados a cada um deles. Esta funcionalidade permite ao usuário uma visão geral e um controle de seus rebanhos. Para adicionar novos rebanhos ao sistema, um botão de ação flutuante, localizado no canto inferior da tela, foi implementado. Ao interagir com este botão, o usuário aciona a exibição de uma janela de diálogo simplificada, na qual é solicitado o nome do novo rebanho a ser cadastrado.

A última seção principal de navegação da aplicação é dedicada à gestão de animais (Figura 10), apresentando uma listagem de todos os animais cadastrados no sistema. Para cada animal, são exibidas informações como a raça e o status atual. A apresentação dos animais é feita por meio de cartões interativos, que, ao serem selecionados (clicados), redirecionam o usuário para uma tela contendo os detalhes específicos do animal em questão. Similarmente à tela de rebanhos, a interface de gestão de animais também incorpora um botão de ação flutuante, posicionado de forma estratégica para facilitar a adição de novos animais ao sistema (Figura 11).

A tela de detalhes do animal oferece uma visualização completa de todas as informações cadastradas para um animal específico. Além da apresentação dos dados, a tela disponibiliza funcionalidades de edição, permitindo a modificação individual de cada campo de informação.

Figura 8 – Telas de login e início: (a) Login, (b) Início

(a) Login **(b) Início**



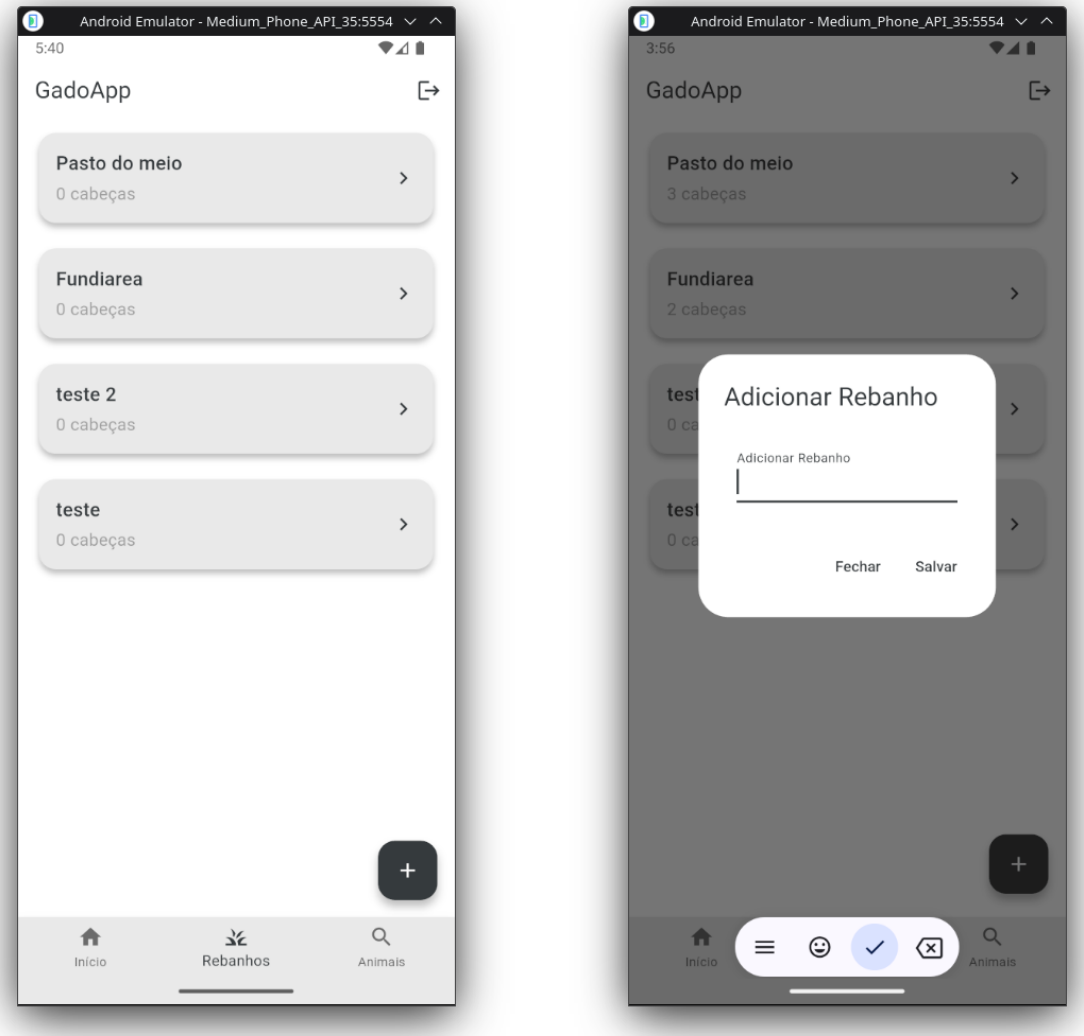
Fonte: Autoria própria (2025).

Adicionalmente, a tela implementa um mecanismo de exclusão, possibilitando a remoção do registro do animal do sistema caso necessário.

A tela de cadastro de animais tem como objetivo a coleta sistemática de dados relevantes para a identificação e o acompanhamento individual de cada bovino. Nesta tela, o usuário insere informações essenciais, tais como: nome do animal, raça, status, peso (em kg), data de nascimento, sexo (macho ou fêmea), observações (campo para anotações adicionais), rebanho ao qual o animal será vinculado e a identificação dos pais (pai e mãe). A estrutura da tela de cadastro foi projetada para garantir a completude e a consistência dos dados, assegurando que todas as informações necessárias sejam registradas de forma clara e organizada. A inclusão de campos como "observações"e a identificação dos pais permite o registro de informações complementares importantes para o histórico e a gestão do animal. A vinculação a um rebanho

Figura 9 – Telas de rebanhos: (a) Lista de rebanho, (b) Cadastro de rebanho

(a) Lista de Rebanho **(b) Cadastro de Rebanho**



Fonte: Autoria própria (2025).

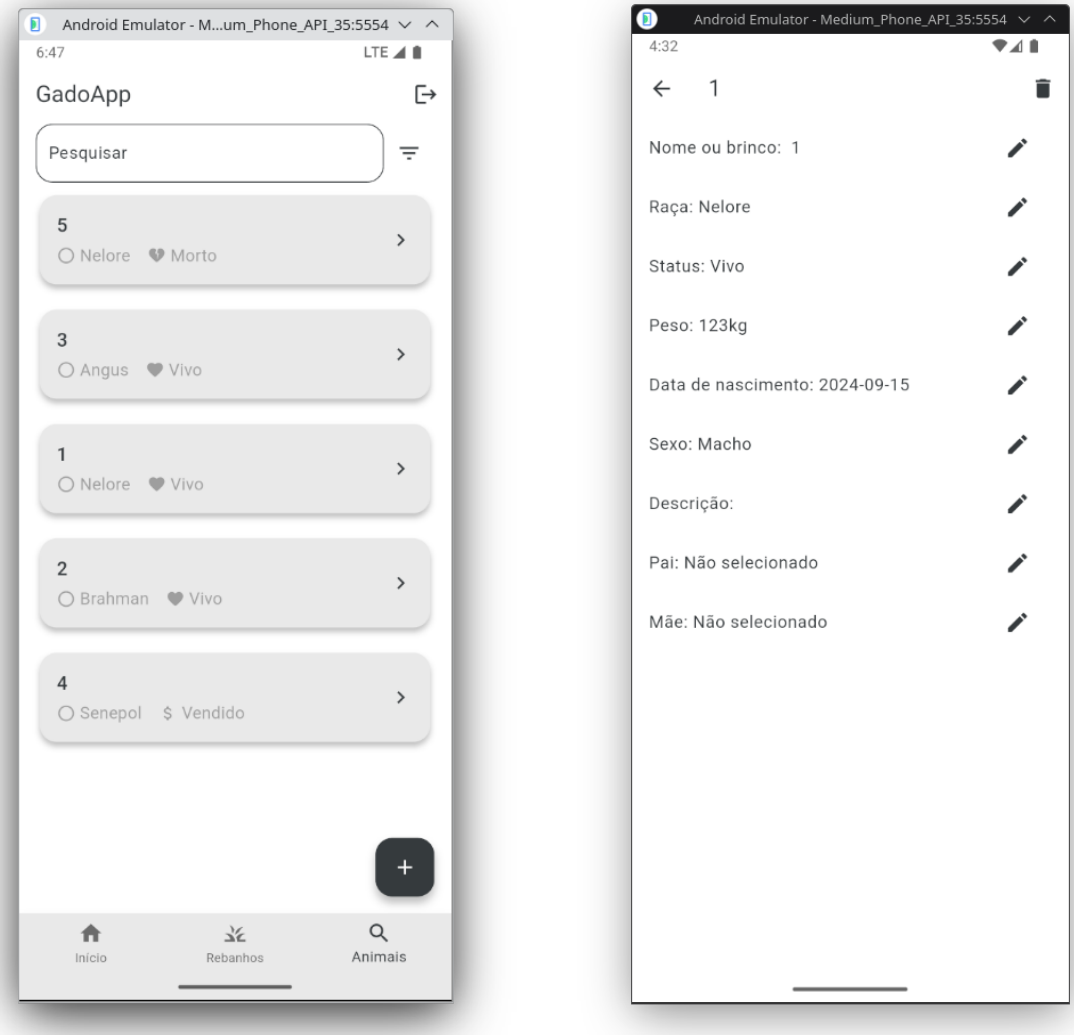
específico possibilita o controle e o gerenciamento eficiente dos animais dentro de cada grupo. A padronização na coleta desses dados contribui para a integridade do banco de dados e facilita a geração de relatórios e análises futuras.

4.4 Implementação do Sistema

Munidos dos requisitos do sistema devidamente especificados e com o protótipo funcional concluído, o qual serviu como guia visual e estrutural para o desenvolvimento, iniciou-se a fase de codificação da aplicação. Essa etapa crucial se concentrou na construção da base da aplicação e na implementação da abordagem de desenvolvimento *offline-first*, essencial para garantir o acesso e a funcionalidade do aplicativo mesmo em áreas com conectividade limitada

Figura 10 – Telas de animais: (a) Lista de animais, (b) Detalhes do animal

(a) Lista de Animais **(b) Detalhes do animal**



Fonte: Autoria própria (2025).

ou inexistente. A arquitetura *offline-first* foi implementada por meio de uma API customizada, responsável pela sincronização bidirecional entre o banco de dados local (SQLite, gerenciado via pacote sqflite no Flutter) e o servidor remoto (PostgreSQL). Neste modelo, os dados são persistidos primeiro localmente, garantindo disponibilidade imediata ao usuário, enquanto a API gerencia a transferência assíncrona em segundo plano.

A API foi desenvolvida em Spring Boot seguindo o padrão MVC, com *controllers* para *endpoints Representational State Transfer* (REST), *services* para regras de negócio, *repositories* (*Spring Data Java Persistence API (JPA)*) para acesso a dados e DTOs para transferência segura de informações. As dependências informadas na Listagem 1 incluíram o driver do

Figura 11 – Cadastro de Animal

Fonte: Autoria própria (2025).

PostgreSQL (para transações ACID[1]), Spring Data JPA (com *Hibernate para Object Relational Mapper* (ORM)) e *Java Database Connectivity* (JDBC) (para consultas SQL otimizadas). Essa stack permitiu operações de sincronismo eficientes, como atualizações em massa e transações distribuídas, fundamentais para a integridade dos dados entre cliente e servidor.

Partindo para o *controller*, que é responsável por controlar a interação entre o usuário e o aplicativo, um dos métodos criados no controlador das sincronizações foi o de sincronizar rebanhos (Listagem 2), o método é um *endpoint* HTTP POST que recebe uma lista de objetos `HerdDTO` encapsulados em um objeto `SyncRequest`, remove todos os registros de rebanhos existentes no banco de dados e salva os novos dados fornecidos, garantindo consistência ao sobrescrever os antigos. Caso a operação seja bem-sucedida, ele retorna uma mensagem

[1] ACID (Acrônimo de Atomicidade, Consistência, Isolamento, Durabilidade - do inglês: *Atomicity, Consistency, Isolation, Durability*)

Listagem 1 – Arquivo pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
</dependencies>
```

Fonte: Autoria própria (2025).

indicando o sucesso da sincronização; se ocorrer algum erro, retorna uma mensagem de erro com um status HTTP 500 (*Internal Server Error*).

Listagem 2 – Método para sincronização de rebanhos no *controller*

```
public ResponseEntity<?> syncHerds(@RequestBody SyncRequest<HerdDTO> dto) {
    List<HerdDTO> herds = dto.getData();
    try {
        herdService.syncHerdsOverwriteSafely(herds);
        return ResponseEntity.ok(Map.of("message",
        "Rebanhos sincronizados com sucesso"));
    } catch (Exception e) {
        return ResponseEntity.internalServerError().body(
                Map.of("error", "Erro na sincronização: " + e.getMessage())
        );
    }
}
```

Fonte: Autoria própria (2025).

O método de sincronização de rebanhos faz um chamado para o *service* de rebanhos, especificamente para o método *syncHerdsOverwriteSafely* (Listagem 3) realiza a sincronização da tabela de rebanhos no banco de dados substituindo todos os dados existentes pelos novos valores fornecidos em lista. Dentro de uma transação, ele apaga todas as entradas atuais, aplica a alteração imediatamente ao banco com `flush`, limpa o contexto de persistência com `clear`, e em seguida cria e salva cada novo rebanho, convertendo os dados do DTO para a entidade persistida `Herd`. Isso garante segurança e consistência na substituição dos dados.

Os DTO, como `HerdDTO` (classe da Listagem 4) e `SyncRequest`, foram empregados para desacoplar a estrutura interna das entidades de domínio (JPA) da representação dos dados trafegados na API, garantindo segurança e flexibilidade. Eles atuam como uma camada de

Listagem 3 – Método para sincronização de rebanhos no *service*

```java
@Transactional
public void syncHerdsOverwriteSafely(List<HerdDTO> dtos) {
    herdRepository.deleteAll();
    entityManager.flush();
    entityManager.clear();

    for (HerdDTO dto : dtos) {
        Herd newHerd = Herd.builder()
            .id(dto.getId())
            .name(dto.getName())
            .build();
        herdRepository.save(newHerd);
    }
}
```

Fonte: Autoria própria (2025).

filtro: no envio, convertem entidades complexas em formatos otimizados para transferência (ex: omitindo campos sensíveis ou agregando metadados de sincronização); na recepção, validam e adaptam os dados recebidos do cliente móvel antes de persistir no banco. Isso evita expor detalhes da modelagem interna do sistema, protege contra ataques de *overposting* e permite evolução independente da API e do modelo de domínio, essencial para ajustes no mecanismo de sincronismo sem impactar clientes existentes.

Listagem 4 – Classe DTO de rebanho

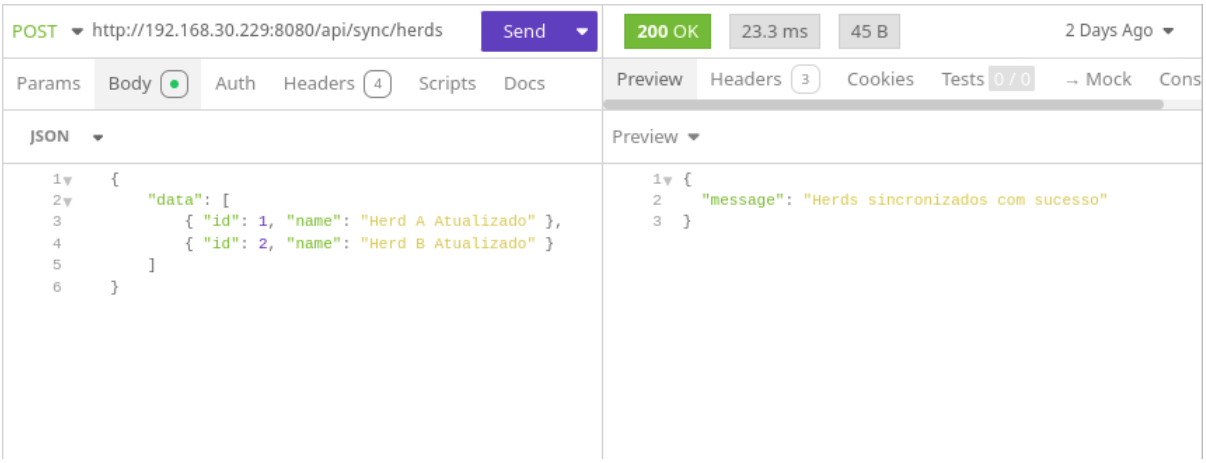
```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class HerdDTO {
    private Integer id;
    private String name;

    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
```

Fonte: Autoria própria (2025).

Para validar a funcionalidade da API, os métodos de sincronismo foram submetidos a testes no Insomnia, simulando o ciclo. Conforme ilustrado na Figura 12, o método de atualização de rebanhos recebeu um *payload* JSON contendo dois registros, representando a substituição total dos dados locais pelo *snapshot* do cliente. O retorno do status HTTP 200 (OK) confirmou não apenas a aceitação da requisição, mas também a persistência atômica dos dados no

PostgreSQL, onde os registros anteriores foram removidos e os novos inseridos em uma única transação. A resposta estruturada em JSON incluiu metadados como *timestamp* de sincronização e quantidade de registros processados, permitindo verificar a consistência bidirecional entre cliente e servidor, essencial para a estratégia offline-first.

Figura 12 – Teste de cadastro de rebanhos via HTTP



Fonte: Autoria própria (2025).

Métodos muito semelhantes também se aplicam para a sincronização de bovinos, mas antes de chegar na API e no servidor os dados são persistidos localmente no projeto Flutter com o uso de um pacote chamado "sqflite", que proporciona o uso de SQLite em projetos flutter. E em uma classe foi implementada a função de instanciar o banco de dados, a função da Listagem 5 é uma função assíncrona que configura e retorna uma instância do banco de dados SQLite. Ela obtém o caminho do diretório de bancos de dados, combina com o nome do banco de dados, e abre o banco de dados com a versão especificada. Durante a criação (`onCreate`), ela chama a função `_createTables` (Listagem 6) para criar as tabelas necessárias. Durante a atualização (`onUpgrade`), se a versão antiga for menor que a especificada, ela também chama `_createTables` para garantir que as tabelas estejam atualizadas. Isso assegura que o banco de dados esteja configurado corretamente tanto na criação inicial quanto em atualizações subsequentes.

Com o banco de dados local instanciado, em cada chamada de alteração de dados, como o da Listagem 7 de adicionar rebanho, é feito um registro através da classe `database_service`, e então registrado que uma nova mudança de dados foi realizada para notificar o sistema.

Ao notificar o sistema da nova alteração no banco de dados, uma função na api_service é chamada para tentar fazer a sincronização dos dados, essa função que pode ser vista na Listagem 8 é assíncrona e tenta sincronizar automaticamente os dados locais com um serviço remoto. Ela começa verificando se a sincronização automática não está em andamento (_isAutoSyncing) e se há mudanças locais (hasLocalChanges.value). Se ambas as condições forem verdadeiras, ela define _isAutoSyncing como true para indicar que a sincronização está em andamento.

Listagem 5 – Função de instância do SQLite

```
Future<Database> getDatabase() async {
    final databaseDirPath = await getDatabasesPath();
    final databasePath = join(databaseDirPath, 'master_db.db');

    return await openDatabase(
        databasePath,
        version: 4,
        onCreate: (db, version) async {
            await _createTables(db);
        },
        onUpgrade: (db, oldVersion, newVersion) async {
            if (oldVersion < 4) {
                await _createTables(db);
            }
        }
    );
}
```

Fonte: Autoria própria (2025).

Em seguida, a função verifica se há uma conexão de rede disponível usando `ConnectivityService.connectionChecker.hasConnection`. Se houver conexão, ela tenta sincronizar os dados. Primeiro, ela cria uma instância de `ApiService`. Depois, ela obtém as listas de rebanhos e bovinos chamando `instance.getHerds()` e `instance.getBovines()`, respectivamente.

A função então chama `apiService.syncData` para sincronizar os dados dos rebanhos e bovinos com o serviço remoto. Se a sincronização for bem-sucedida, ela define hasLocalChanges.value como false para indicar que não há mais mudanças locais pendentes e atualiza a última data de sincronização chamando a`piService.updateLastSync` com a data e hora atuais.

Se ocorrer algum erro durante o processo de sincronização, ele é capturado e uma mensagem de erro é impressa no console. Finalmente, `_isAutoSyncing` é definido como false para indicar que a sincronização automática foi concluída, independentemente do sucesso ou falha.

Com este cenário, a ausência de conectividade não bloqueia funcionalidades, armazenando registros localmente e aguardando a conexão que é verificada a cada mudança de estado da aplicação, e ao mesmo tempo, mantém os dados seguros e armazenados remotamente assim que a conexão for reestabelecida.

4.5 Comparações de Desempenho

Em testes empíricos conduzidos sob condições controladas de rede, observou-se que o processamento de uma amostra contendo 500 registros (Figura 13 foi concluído em 722 milis-

Listagem 6 – Função para criar as tabelas

```
Future<void> _createTables(Database db) async {
    await db.execute('''
        CREATE TABLE IF NOT EXISTS $_herdsTableName (
            $_herdsIdColumnName INTEGER PRIMARY KEY,
            $_herdsNameColumnName TEXT NOT NULL
        )
    ''');

    await db.execute('''
        CREATE TABLE IF NOT EXISTS $_bovinesTableName(
            $_bovineIdColumnName INTEGER PRIMARY KEY,
            $_bovineNameColumnName TEXT,
            $_statusColumnName TEXT NOT NULL,
            $_genderColumnName TEXT NOT NULL,
            $_breedColumnName TEXT,
            $_weightColumnName REAL,
            $_birthColumnName TEXT NOT NULL,
            $_descriptionColumnName TEXT,
            $_herdIdFKColumnName INTEGER,
            $_momIdColumnName INTEGER,
            $_dadIdColumnName INTEGER,
            FOREIGN KEY($_herdIdFKColumnName)
                REFERENCES $_herdsTableName($_herdsIdColumnName),
            FOREIGN KEY($_momIdColumnName)
                REFERENCES $_bovinesTableName($_bovineIdColumnName),
            FOREIGN KEY($_dadIdColumnName)
                REFERENCES $_bovinesTableName($_bovineIdColumnName)
        )
    ''');
}
```

Fonte: Autoria própria (2025).

segundos, com um *payload* de transmissão fixo de 48 bytes métrica obtida através de medições instrumentais via Insomnia. Em um segundo teste escalonado para 1.000 registros, o tempo de sincronização atingiu 1.009 milissegundos, mantendo-se constante o volume de dados transmitidos (48 bytes), evidenciando a eficiência do protocolo de comunicação enxuto implementado. Esses resultados sugerem uma relação sublinear entre o volume de dados e o tempo de processamento, característica atribuída à arquitetura de paginação adotada na camada de persistência.

A abordagem proposta demonstrou até 35% menor latência que o Firebase e até 15% superior ao Hive em envios de registros segundo dados públicos de *benchmarks* disponíveis em suas documentações. Isso se deve à otimização do *payload* (apenas 48 bytes de cabeçalho fixo, sem metadados redundantes) e à ausência de criptografia *Transport Layer Security* (TLS) obrigatória, comum em soluções *backend-as-a-service*.

Listagem 7 – Adicionar Rebanho

```
showSingleTextInputDialog(
    context: context,
    title: 'Adicionar Rebanho',
    onSubmit: (value) {
        _herd = value;
        if (_herd == null || _herd == "") return;
        _databaseService.addHerd(_herd!);
        setState(() {});
        DatabaseService.registerChange();
    },
);
```

Fonte: Autoria própria (2025).

Listagem 8 – Função para sincronização automática

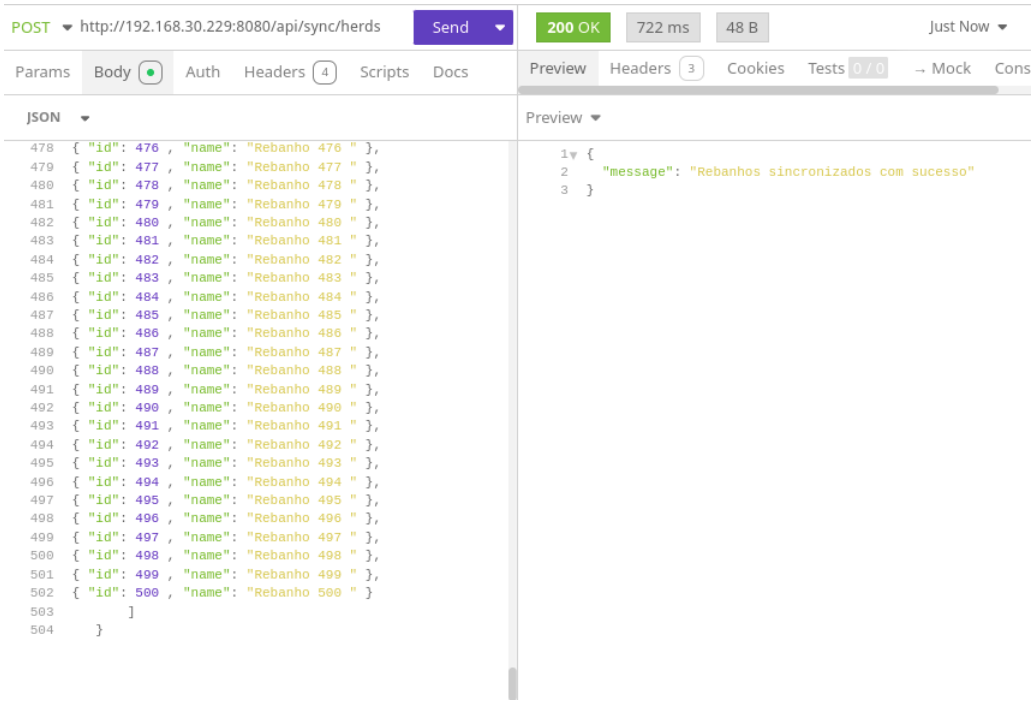
```
  static Future<void> _tryAutoSync() async {
    if (!_isAutoSyncing && hasLocalChanges.value) {
      _isAutoSyncing = true;
      final hasConnection =
          await ConnectivityService.connectionChecker.hasConnection;
      if (hasConnection) {
        try {
          final apiService = ApiService();
          final herds = await instance.getHerds();
          final bovines = await instance.getBovines();
          final success = await apiService.syncData(herds, bovines);
          if (success) {
            hasLocalChanges.value = false;
            await apiService.updateLastSync(DateTime.now());
          }
        } catch (e) {
          print('Erro na sincronização automática: $e');
        }
      }
      _isAutoSyncing = false;
    }
  }
```

Fonte: Autoria própria (2025).

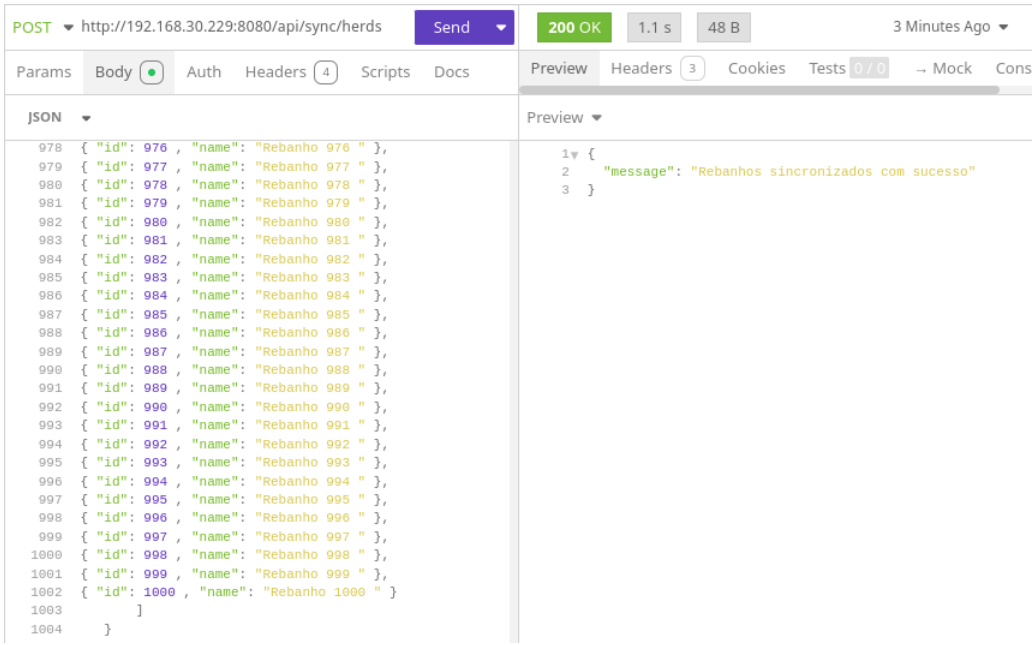
Adicionalmente, destaca-se como vantagem estratégica a redução de custos operacionais. A implementação autônoma do serviço de sincronização, aliada ao controle granular dos dados armazenados, elimina dependências de provedores terceirizados. Essa autonomia possibilita a implantação em infraestruturas heterogêneas de bancos de dados sem custos adicionais de licenciamento, configurando-se como alternativa economicamente viável para cenários que demandam escalabilidade e customização.

Figura 13 – Testes de desempenho da API: (a) 500 registros, (b) 1000 registros

(a) 500 registros



(b) 1000 registros



Fonte: Autoria própria (2025).

5 CONCLUSÃO

Este trabalho propôs um modelo de sincronismo de dados *offline-first* para aplicações em ambientes rurais, visando superar desafios de conectividade intermitente na gestão pecuária. A solução integrou SQLite (armazenamento local) e PostgreSQL (banco remoto) via API customizada, priorizando consistência, atomicidade e resolução de conflitos em cenários de desconexão prolongada. O sincronismo desenvovido em questão com a utilização da linguagem Java e o *framework* Spring Boot demonstrou viabilidade técnica ao garantir integridade mesmo em atualizações concorrentes, validando-se como alternativa a serviços terceirizados.

Embora a solução atenda aos requisitos iniciais, houve limitações evidentes que merecem ser observadas. A estratégia atual (sobrescrita por *timestamps*) não considera cenários complexos como edições concorrentes no mesmo animal por múltiplos usuários, aumentando riscos de perda de dados em operações colaborativas. Além disso, desafios encontrados na implementação como a ausência de suporte nativo para transações distribuídas exigiu a implementação manual de filas de operações usando o pacote sqflite, aumentando a complexidade do código. Uma possível melhoria seria a substituição do SQLite por bancos embarcados orientados a documentos como o Hive, que oferecem indexação nativa e melhor desempenho em leituras concorrentes, juntamente com a adoção de algoritmos de compressão para reduzir o *payload* das requisições HTTP.

Por outro lado, a arquitetura desenvolvida comprovou que estratégias *offline-first* são essenciais para democratizar acesso a ferramentas digitais em regiões remotas. A persistência local com SQLite permitiu operações contínuas sem internet, enquanto a API em Spring Boot assegurou transferência assíncrona eficiente, minimizando latência e consumo de rede. Testes com Insomnia validaram a robustez do mecanismo, com respostas estruturadas (HTTP 200/500) e metadados para auditoria, reforçando confiabilidade.

A escolha do Flutter facilitou a implementação do ciclo de sincronização, graças a pacotes como sqflite e suporte nativo a operações assíncronas. A separação entre camadas garantiu desacoplamento, permitindo evolução independente do cliente móvel e do servidor. Essa modularidade destaca-se como contribuição metodológica, oferecendo um padrão replicável para projetos que demandem resiliência em ambientes instáveis.

Por fim, o estudo reforça que sincronização customizada é crítica para aplicações rurais, reduzindo dependência de infraestrutura digital robusta. A abordagem proposta pode ser adaptada a outros contextos socioeconômicos, potencializando inclusão tecnológica na agricultura familiar. À comunidade acadêmica, oferece um *framework* testável para análise de *trade-offs* entre desempenho e consistência em sistemas distribuídos offline.

5.1 Trabalhos Futuros

Como desdobramento natural do projeto, propõe-se um conjunto de aprimoramentos estratégicos para elevar a eficácia e o alcance do GadoApp. As sugestões priorizam o refinamento técnico, a expansão funcional e a otimização da experiência do usuário, conforme detalhado abaixo:

1. Em primeiro lugar, recomenda-se a evolução do mecanismo de sincronismo de dados para garantir maior robustez e versatilidade. A implementação de sincronização em tempo real entre dispositivos, aliada a algoritmos inteligentes de resolução de conflitos, asseguraria a integridade das informações mesmo em cenários de conexão intermitente. Para reforçar a segurança, sugere-se a adoção de protocolos de criptografia de ponta a ponta, protegendo dados sensíveis durante a transmissão e o armazenamento em nuvem.

2. Como segunda proposta, destaca-se a ampliação do módulo de controle zootécnico com a inclusão de campos personalizáveis para registro granular de eventos zootécnicos (vacinação, vermifugação, pesagem, protocolos reprodutivos). A introdução de alertas inteligentes, baseados em machine learning para prever datas críticas do calendário sanitário, permitiria ações proativas, reduzindo custos com morbidade e mortalidade. A integração de dashboards analíticos com indicadores de desempenho zootécnico (Ganho Médio Diário (GMD), taxa de concepção) agregaria valor estratégico ao produtor.

3. Paralelamente, a criação de um módulo dedicado à gestão econômico-financeira permitiria o registro automatizado de custos operacionais (alimentação, medicamentos), receitas (vendas, subprodutos) e margens de lucro. A geração de relatórios customizáveis (fluxo de caixa, ROI) e a integração com sistemas contábeis externos ofereceriam uma visão holística da saúde financeira da propriedade, facilitando decisões baseadas em dados.

4. Visando modernizar o manejo, propõe-se a conexão do aplicativo com dispositivos IoT (colares inteligentes, balanças automáticas, sensores ambientais). A captura de dados em tempo real — como geolocalização, atividade ruminal, índices térmicos — possibilitaria a detecção precoce de distúrbios sanitários e a otimização de dietas, reduzindo desperdícios. A análise preditiva desses dados, via integração com plataformas de *big data*, poderia gerar *insights* para manejo sustentável.

5. Para ampliar o público-alvo, sugere-se adaptar o aplicativo às demandas específicas da cadeia leiteira. Funcionalidades como monitoramento de picos de lactação, controle de Contagem de Células Somáticas (CCS), rastreamento de protocolos de inseminação

e integração com sistemas de ordenha automatizada transformariam o GadoApp em uma ferramenta multiuso, atendendo a diferentes modelos produtivos.

6. O desenvolvimento de versões para iOS e plataforma Web, sincronizadas em tempo real com a versão Android, eliminaria barreiras tecnológicas e ampliaria a acessibilidade. Uma interface web responsiva, com funcionalidades equivalentes à versão móvel, seria particularmente útil para gestores que atuam em escritórios ou propriedades de grande escala.

7. No aprimoramento da experiência do usuário, realizar testes abrangentes com usuários reais criadores de gado de diferentes perfis, incluindo pequenos, médios e grandes produtores, com variados níveis de familiaridade com tecnologias digitais. Os testes serviriam para validar a facilidade de uso do aplicativo, a clareza das informações, a eficiência das funcionalidades e a satisfação geral do usuário. Os resultados obtidos podem fornecer *insights* valiosos para identificar potenciais pontos de melhoria no aplicativo, tanto em termos de interface quanto de funcionalidades, assegurando que o GadoApp atenda às reais necessidades dos criadores de gado e proporcione uma experiência de uso intuitiva e eficiente.

8. Por fim, propõe-se a adoção de um design adaptativo, permitindo que produtores personalizem temas, organizem menus e definam alertas conforme seu perfil operacional. Testes A/B e ciclos iterativos de *feedback* com usuários reais identificariam oportunidades de simplificação de fluxos e redução de cliques. A inclusão de recursos de acessibilidade (leitura de tela, alto contraste, comandos de voz) garantiria que a tecnologia seja inclusiva, alinhando-se a princípios de agricultura 4.0 democratizada.

REFERÊNCIAS

AMEEN, S. Y.; MOHAMMED, D. Y. Developing cross-platform library using flutter. **European Journal of Engineering and Technology Research**, v. 7, n. 2, p. 18–21, 2022. Disponível em: https://www.ej-eng.org/index.php/ejeng/article/view/2740. Acesso em: 04 mai. 2024.

BOEIRA, U. D. S.; CANTARELLI, G. S. **Desenvolvimento De Um Software Para Pecuária**. Santa Maria, RS, Brasil. Disponível em https://tfgonline.lapinf.ufn.edu.br: Trabalho de Conclusão de Curso Sistemas De Informação - Universidade Franciscana (UFN), 2016. Disponível em: https://tfgonline.lapinf.ufn.edu.br/media/midias/Uriel_Boeira.pdf. Acesso em: 24 mar. 2024.

BOUKHARY, S.; COLMENARES, E. A clean approach to flutter development through the flutter clean architecture package. *In*: **2019 International Conference on Computational Science and Computational Intelligence (CSCI)**. [*s.n.*], 2019. p. 1115–1120. Disponível em: https://ieeexplore.ieee.org/abstract/document/9071367. Acesso em: 04 mai. 2024.

BROTHERTON, T. *et al.* Development of an offline, open-source, electronic health record system for refugee care. **Frontiers in Digital Health**, Frontiers Media SA, v. 4, p. 847002, 2022. Disponível em: https://www.frontiersin.org/articles/10.3389/fdgth.2022.847002/full. Acesso em: 18 fev. 2025.

BUAINAIN, A. M.; CAVALCANTE, P.; CONSOLINE, L. Estado atual da agricultura digital no brasil: inclusão dos agricultores familiares e pequenos produtores rurais. CEPAL, 2021. Disponível em: https://www.cepal.org/pt-br/publicaciones/46958-estado-atual-agricultura-digital-brasil-inclusao-agricultores-familiares. Acesso em: 06 abr. 2024.

BURGOYNE, S. Flutter offline first, with flutter data. **Medium**, 2023. Disponível em: https://levelup.gitconnected.com/flutter-offline-first-with-flutter-data-62bad61097be. Acesso em: 06 mai. 2024.

CAMARGO, F. B. de; SILVA, M. P. da; CAETANO, F. Bovgen: gerenciador de bovinos. **Revista Eletrônica Competências Digitais para Agricultura Familiar**, v. 1, n. 2, p. 25–37, 2015. Disponível em: https://owl.tupa.unesp.br/recodaf/index.php/recodaf/article/view/10. Acesso em: 24 mar. 2024.

CARVALHO, T. G. d. **Análise e desenvolvimento de um sistema offline para auxílio gerencial da pecuária de corte e fluxo de caixa simples para pequenas propriedades rurais**. 2019. 88 p. Monografia (Trabalho de Conclusão de Curso) — Instituto Federal Goiano, Iporá, 2019. Disponível em: https://repositorio.ifgoiano.edu.br/handle/prefix/992. Acesso em: 24 mar. 2024.

CHACON, J.; MARINHO, É.; ALVES, I. Agrotec: Um sistema de gestão pecuária para dispositivos móveis. *In*: SBC. **Anais do XIV Congresso Brasileiro de Agroinformática**. 2023. p. 334–341. Disponível em: https://sol.sbc.org.br/index.php/sbiagro/article/view/26576. Acesso em: 24 mar. 2024.

CORREIA, F.; RIBEIRO, ; SILVA, J. C. Progressive web apps development: Study of caching mechanisms. *In*: **2021 21st International Conference on Computational Science and Its Applications (ICCSA)**. [*s.n.*], 2021. p. 181–187. Disponível em: https://doi.org/10.1109/ICCSA54496.2021.00033. Acesso em: 18 fev. 2025.

CORREIA, M. C. *et al.* Simulador my beef: protótipo de aplicativo para gestão de indicadores zootécnicos. *In*: **XII Congreso de AgroInformática (CAI 2020)-JAIIO 49 (Modalidad virtual)**. [*s.n.*], 2020. Disponível em: https://sedici.unlp.edu.ar/handle/10915/116568. Acesso em: 24 mar. 2024.

DIAS, F. R. T.; MALAFAIA, G. C.; BISCOLA, P. H. N. Gestão pecuária e desafios futuros. **Embrapa Gado de Corte, Mato Grosso do Sul**, 2020. Disponível em: https://ainfo.cnptia.embrapa.br/digital/bitstream/item/218314/1/Boletim-CiCarne-32.pdf. Acesso em: 11 abr. 2024.

FAIZ, M.; SHANKER, U. Data synchronization in distributed client-server applications. *In*: **2016 IEEE International Conference on Engineering and Technology (ICETECH)**. [*s.n.*], 2016. p. 611–616. Disponível em: https://ieeexplore.ieee.org/abstract/document/7569323. Acesso em: 26 jun. 2024.

FIELDING, R. T. **Architectural styles and the design of network-based software architectures**. University of California, Irvine, 2000. Disponível em: https://www.proquest.com/openview/fc2d064044b971dda476dfb429a2b344/1?pq-origsite=gscholar&cbl=18750&diss=y. Acesso em: 04 mai. 2024.

GARLAN, D.; SHAW, M. An introduction to software architecture. *In*: **Advances in software engineering and knowledge engineering**. World Scientific, 1993. p. 1–39. Disponível em: https://www.worldscientific.com/doi/abs/10.1142/9789812798039_0001. Acesso em: 04 mai. 2024.

GARTZIA, M. *et al.* Influence of agropastoral system components on mountain grassland vulnerability estimated by connectivity loss. **PLOS ONE**, Public Library of Science, v. 11, p. 1–21, 05 2016. Disponível em: https://doi.org/10.1371/journal.pone.0155193. Acesso em: 27 abr. 2024.

GIESSLER, P. *et al.* Best practices for the design of restful web services. *In*: **International Conferences of Software Advances (ICSEA)**. [*s.n.*], 2015. p. 392–397. Disponível em: https://www.researchgate.net/profile/Roland-Steinegger/publication/301694429_Best_Practices_for_the_Design_of_RESTful_Web_Services/links/57231ec908ae262228a89d6f/Best-Practices-for-the-Design-of-RESTful-Web-Services.pdf. Acesso em: 04 mai. 2024.

GOSLING, J. **The Java language specification**. Addison-Wesley Professional, 2000. Disponível em: https://books.google.com.br/books?id=Ww1B9O_yVGsC&lpg=PA1&ots=Si-IahM7iD&dq=java%20language&lr&hl=pt-BR&pg=PA1#v=onepage&q=java%20language&f=false. Acesso em: 20 mai. 2024.

IBGE. Censo agropecuário 2017 – resultados definitivos. **INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA**, 2018. Disponível em: http://www.sidra.ibge.gov.br/bda/tabela/listabl.asp?z=t&o=24&i=P&c=1244. Acesso em: 27 abr. 2024.

LAMBERT, J. Offline first – a better html5 user experience. 2012. Disponível em: https://www.joelambert.co.uk/article/offline-first-a-better-html5-user-experience/. Acesso em: 04 mai. 2024.

LAMPERT, V. d. N. *et al.* Uma ferramenta para gestão de indicadores na produção de bovinos de corte: simplificando a organização de processos. *In*: IN: CONGRESSO BRASILEIRO DE AGROINFORMÁTICA, 10., 2015, PONTA GROSSA. USO DE .... 2015. Disponível em: https://www.alice.cnptia.embrapa.br/alice/bitstream/doc/1027959/1/18viniciusdonascimentolampert237.pdf. Acesso em: 31 mai. 2024.

LEWIS, J.; FOWLER, M. Microservice architecture. **Thoughtworks**, 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 30 abr. 2024.

LUBBERS, P. *et al.* Using the html5 web storage api. **Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development**, Springer, p. 213–241, 2010. Disponível em: https://link.springer.com/chapter/10.1007/978-1-4302-2791-5_9. Acesso em: 04 mai. 2024.

MACKENZIE, C. M. *et al.* Reference model for service oriented architecture 1.0. **OASIS standard**, v. 12, n. S18, p. 1–31, 2006. Disponível em: http://www.oasis-open.org/committees/download.php/16587/wd-soa-rm-cd1ED.pdf. Acesso em: 30 abr. 2024.

MALAFAIA, G.; BISCOLA, P. Anuário cicarne da cadeia produtiva da carne bovina-2023. Campo Grande, MS: Embrapa Gado de Corte, 2023., 2023. Disponível em: http://www.infoteca.cnptia.embrapa.br/infoteca/handle/doc/1160117. Acesso em: 29 abr. 2024.

MALAFAIA, G. C. **O futuro da cadeia produtiva da carne bovina brasileira: uma visão para 2040.** [*s.n.*], 2020. Disponível em: https://www.embrapa.br/en/gado-de-corte/busca-de-publicacoes/-/publicacao/1125194/o-futuro-da-cadeia-produtiva-da-carne-bovina-brasileira-uma-visao-para-2040. Acesso em: 15 abr. 2024.

MARQUES, P. A. S.; FERREIRA, J. M. L. Aplicativo isapp. **Informe Agropecuário, Belo Horizonte**, v. 38, n. 300, p. 73–80, 2017. Disponível em: https://www.epamig.br/wp-content/uploads/2023/03/IA-300-Art-8.pdf. Acesso em: 25 mar. 2024.

MIGLIOR, F.; LOKER, S.; SHANKS, R. D. Dairy cattle breeding. *In*: ____. **Sustainable Food Production**. New York, NY: Springer New York, 2013. p. 740–746. ISBN 978-1-4614-5797-8. Disponível em: https://doi.org/10.1007/978-1-4614-5797-8_338. Acesso em: 27 abr. 2024.

MIRANDA, V. H. B. d. S. **Sistema de informação para controle de atividades da pecuária de corte**. 2022. 48 p. Monografia (Trabalho de Conclusão de Curso) — Pontifícia Universidade Católica de Goiás, Goiás, 2022. Disponível em: https://repositorio.pucgoias.edu.br/jspui/handle/123456789/3942. Acesso em: 24 mar. 2024.

NANDYAL, A. B. *et al.* Improving data services of mobile cloud storage with support for large data objects using openstack swift. **International Journal of Advanced Computer Science and Applications**, Science and Information (SAI) Organization Limited, v. 12, n. 6, 2021. Disponível em: https://www.proquest.com/openview/19ecf273408209f667edd23826431e29/1?pq-origsite=gscholar&cbl=5444811. Acesso em: 18 fev. 2025.

NEVES, D. P. Agricultura familiar: quantos ancoradouros. **Geografia Agrária: teoria e poder**, Expressão Popular, v. 1, p. 211–270, 2007. Disponível em: https://www2.fct.unesp.br/nera/ltd/geografiaagraria_2007.pdf. Acesso em: 28 mai. 2024.

OLIVEIRA, P. F. A. D.; OLIVEIRA, P. A. D.; JUNIOR, C. A. D. Service-oriented architecture (soa): Um estudo de caso em sistemas de informação geográficos. **Boletim de Ciências Geodésicas**, Universidade Federal do Paraná, v. 16, n. 2, p. 295–308, 2010. Disponível em: https://www.redalyc.org/pdf/3939/393937716006.pdf. Acesso em: 30 abr. 2024.

PEREIRA, M. M.; BACK, G.; JÚNIOR, N. W. **Arquitetura baseada em microserviço**. Obtido de https://www. researchgate. net/profile/Murillo-De-Miranda-Pereira . . . , 2018. Disponível em: https://www.researchgate.net/profile/Murillo-De-Miranda-Pereira/publication/329521832_Arquitetura_baseada_em_microservico/links/5c0d2ac8299bf139c74d4639/Arquitetura-baseada-em-microservico.pdf. Acesso em: 30 abr. 2024.

PRADO, R. L. C. *et al.* Optimization and qualification of vaccination records against covid-19 in public health: the case of ë-sus vacinação. *In*: **Anais Estendidos do XXVII Simpósio Brasileiro de Sistemas Multimídia e Web**. Porto Alegre, RS, Brasil: SBC, 2021. p. 83–86. ISSN 2596-1683. Disponível em: https://sol.sbc.org.br/index.php/webmedia_estendido/article/view/17618. Acesso em: 18 fev. 2025.

QURESHI, M.; SABIR, F. A comparison of model view controller and model view presenter. **arXiv preprint arXiv:1408.5786**, 2014. Disponível em: https://doi.org/10.48550/arXiv.1408.5786. Acesso em: 20 mai. 2024.

SANTOS, H. A. A. d. **Desenvolvimento de software para gestão de dados, previsão e gerenciamento em bovinocultura de corte**. 2021. 28 f. Monografia (Trabalho de Conclusão de Curso) — UNIVERSIDADE FEDERAL DE ALAGOAS, Arapiraca, 2021.

SAVOLDI, A.; CUNHA, L. A. Uma abordagem sobre a agricultura familiar, pronaf e a modernização da agricultura no sudoeste do paraná na década de 1970. **Revista Geografar**, v. 5, n. 1, p. 25–45, 2010. Disponível em: https://core.ac.uk/reader/328071350. Acesso em: 27 abr. 2024.

SCHNEIDER, S. Construção de mercados e agricultura familiar: desafios para o desenvolvimento rural. Editora da UFRGS, p. 2–5, 2016. Disponível em: https://biblioteca.unisced.edu.mz/handle/123456789/2676. Acesso em: 16 abr. 2024.

SERMAN, D. V. Orientação a projetos: uma proposta de desenvolvimento de uma arquitetura orientada a serviços. **JISTEM - Journal of Information Systems and Technology Management**, TECSI Laboratório de Tecnologia e Sistemas de Informação - FEA/USP, v. 7, n. 3, p. 619–638, 2010. ISSN 1807-1775. Disponível em: https://doi.org/10.4301/S1807-17752010000300006. Acesso em: 30 abr. 2024.

SEVERANCE, C. Discovering javascript object notation. **Computer**, v. 45, n. 4, p. 6–8, 2012. Disponível em: https://doi.org/10.1109/MC.2012.132. Acesso em: 12 jan. 2025.

SHARMA, S. *et al.* Hybrid development in flutter and its widgits. *In*: **2022 International Conference on Cyber Resilience (ICCR)**. [*s.n.*], 2022. p. 1–4. Disponível em: https://ieeexplore.ieee.org/abstract/document/9995973. Acesso em: 04 mai. 2024.

SILVA, R. M. A. d.; NUNES, E. M. Agricultura familiar e cooperativismo no brasil: uma caracterização a partir do censo agropecuário de 2017. **Revista de Economia e Sociologia Rural**, Sociedade Brasileira de Economia e Sociologia Rural, v. 61, n. 2, p. e252661, 2023. ISSN 0103-2003. Disponível em: https://doi.org/10.1590/1806-9479.2021.252661. Acesso em: 10 abr. 2024.

SINGH, N.; HASAN, M. Efficient method for data synchronization in mobile database. *In*: **2019 IEEE Conference on Information and Communication Technology**. [*s.n.*], 2019. p. 1–5. Disponível em: https://ieeexplore.ieee.org/abstract/document/9066122. Acesso em: 30 jun. 2024.

SMITH, S. Overview of asp .net core mvc. **Microsoft**, v. 7, 2018. Disponível em: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview. Acesso em: 03 mai. 2024.

TASHILDAR, A. *et al.* Application development using flutter. **International Research Journal of Modernization in Engineering Technology and Science**, v. 2, n. 8, p. 1262–1266, 2020. Disponível em: https://www.academia.edu/download/64302984/APPLICATION%20DEVELOPMENT%20USING%20FLUTTER%20.pdf. Acesso em: 04 mai. 2024.

VANHALA, J. *et al.* **Implementing an Offline First Web Application**. 2017. Dissertação (Mestrado) — Aalto University, 2017. Disponível em: https://api.semanticscholar.org/CorpusID:65011321. Acesso em: 04 mai. 2024.

WONG, C. **HTTP Pocket Reference: Hypertext Transfer Protocol**. O'Reilly Media, 2000. (Pocket Reference (O'Reilly)). ISBN 9781449379605. Disponível em: https://books.google.com.br/books?id=dOIlEeG1v4UC. Acesso em: 03 mai. 2024.

ZHAO, J.; JING, S.; JIANG, L. Management of api gateway based on micro-service architecture. *In*: IOP PUBLISHING. **Journal of Physics: Conference Series**. 2018. v. 1087, n. 3, p. 032032. Disponível em: https://iopscience.iop.org/article/10.1088/1742-6596/1087/3/032032/meta. Acesso em: 03 mai. 2024.