UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

JOÃO GABRIEL JARUTAIS

APLICAÇÃO WEB PARA GESTÃO DE PEQUENAS PROPRIEDADES RURAIS

PATO BRANCO

2025

JOÃO GABRIEL JARUTAIS

APLICAÇÃO WEB PARA GESTÃO DE PEQUENAS PROPRIEDADES RURAIS

WEB APPLICATION FOR SMALL RURAL PROPERTY MANAGEMENT

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Tecnólogo em Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientadora: Andreia Scariot Beulke

PATO BRANCO
2025

JOÃO GABRIEL JARUTAIS

APLICAÇÃO WEB PARA GESTÃO DE PEQUENAS PROPRIEDADES RURAIS

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 26/ novembro/ 2025

Andréia Scariot Beulke
Mestrado
Universidade Tecnológica Federal do Paraná - Campus Pato Branco

---

Lucilia Yoshie Araki
Mestrado
Universidade Tecnológica Federal do Paraná - Campus Pato Branco

---

Mariza Miola Dosciatti
Doutora
Universidade Tecnológica Federal do Paraná - Campus Pato Branco

PATO BRANCO

2025

RESUMO

A agricultura familiar exerce um papel essencial na produção de alimentos e no desenvolvimento econômico das áreas rurais. Contudo, muitos pequenos produtores ainda realizam o controle de suas atividades de forma manual, utilizando cadernos e anotações, o que dificulta a organização financeira e operacional das propriedades. O sistema de gestão rural proposto neste trabalho tem como objetivo oferecer uma ferramenta digital integrada que auxilie no controle financeiro, na gestão de estoque e no acompanhamento do rebanho, substituindo práticas manuais por um ambiente unificado e automatizado. O sistema permite cadastro de categorias financeiras, registro de contas a pagar e a receber, controle de entradas e saídas de insumos, além do gerenciamento das gestações e previsões de partos do rebanho bovino. Também foi implementada uma funcionalidade de envio automático de notificações via WhatsApp, que alerta o usuário sobre vencimentos de contas e datas importantes relacionadas à pecuária. A aplicação foi desenvolvida no formato *web*, utilizando Java com o framework Spring Boot no *back-end*, React JS com TypeScript no *front-end* e o banco de dados PostgreSQL para armazenamento das informações, garantindo robustez, escalabilidade e facilidade de uso em diferentes dispositivos. Com base nos resultados obtidos, verificou-se que o sistema desenvolvido foi capaz de proporcionar uma solução eficiente para o gerenciamento das operações rurais. As funcionalidades de controle financeiro, de estoque e de gestão pecuária demonstraram ser eficazes na centralização das informações e na automatização de processos anteriormente manuais. A integração entre os módulos permitiu ao usuário obter uma visão ampla da propriedade, facilitando a tomada de decisões e o planejamento das atividades. A utilização de gráficos e relatórios personalizados contribuiu para uma análise mais detalhada dos dados, enquanto o envio de lembretes automáticos otimizou o controle de prazos e eventos. Dessa forma, conclui-se que o sistema proposto representa um avanço significativo na informatização da gestão rural, promovendo maior eficiência e contribuindo para a modernização da agricultura familiar.

Palavras-chave: agricultura familiar; pequenos agricultores; gestão rural; sistemas *web*; React JS.

ABSTRACT

Family farming plays an essential role in food production and in the economic development of rural areas. However, many small producers still manage their activities manually, using notebooks and written records, which makes the financial and operational organization of their properties more difficult. The rural management system proposed in this work aims to offer an integrated digital tool to assist in financial control, inventory management, and livestock monitoring, replacing manual practices with a unified and automated environment. The system allows the registration of financial categories, the recording of accounts payable and receivable, and the control of input entries and outputs, in addition to managing pregnancies and predicting births within the cattle herd. An automatic notification feature via WhatsApp was also implemented to alert users about bill due dates and important livestock-related events. The application was developed in a web format, using Java with the Spring Boot framework on the back end, React JS with TypeScript on the front end, and the PostgreSQL database for information storage, ensuring robustness, scalability, and ease of use across different devices.

Based on the results obtained, it was found that the developed system was able to provide an efficient solution for managing rural operations. The financial, inventory, and livestock management functionalities proved effective in centralizing information and automating previously manual processes. The integration among modules enabled users to obtain a comprehensive view of the property, facilitating decision-making and activity planning. The use of customized charts and reports contributed to a more detailed analysis of data, while the automatic reminder feature optimized the control of deadlines and events. Thus, it is concluded that the proposed system represents a significant advancement in the digitalization of rural management, promoting greater efficiency and contributing to the modernization of family farming.

Keywords: family farming; small farmers; rural management; web systems; React JS.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CSRF	*Cross-Site Request Forgery*
DOM	*Document Object Model*
DTO	*Data Transfer Object*
E2E	*End-to-End*
FAO	Organização das Nações Unidas para a Alimentação e a Agricultura
GPS	*Global Positioning System*
HTTP	*Hypertext Transfer Protocol*
JPA	*Java Persistence API*
JPQL	Java Persistence Query Language
JSON	*JavaScript Object Notation*
JSX	*JavaScript* XML
JVM	*Java Virtual Machine*
LDAP	*Lightweight Directory Access Protocol*
MVC	*Model-View-Controller*
NF-e	Nota Fiscal Eletrônica
OAuth2	*Open Authorization 2.0*
PDF	Portable Document Format
PIB	Produto Interno Bruto
REST	*Representational State Transfer*
SAML	*Security Assertion Markup Language*
SI	Sistemas de Informação
SMTP	*Simple Mail Transfer Protocol*
SQL	*Structured Query Language*
TCC1	Trabalho de Conclusão de Curso 1
TCC2	Trabalho de Conclusão de Curso 2
UML	Linguagem de Modelagem Unificada
XLSX	*Excel Open XML Spreadsheet*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais contendo o contexto deste trabalho, os objetivos pretendidos e a justificativa de realização de acordo com os objetivos.

1.1 Considerações iniciais

A agricultura contribui para a produção de alimentos em escala global, com os pequenos agricultores emergindo como pilares essenciais desse sistema. Segundo dados da Organização das Nações Unidas para a Alimentação e a Agricultura (FAO) em 2021, os pequenos agricultores são responsáveis por aproximadamente um terço da produção mundial de alimentos (FAO, 2021). Além disso, pesquisas da FAO em 2014 revelaram que muitas fazendas ao redor do mundo são de caráter familiar, com cerca de 80% dos alimentos sendo provenientes dessas unidades produtivas (FAO, 2021). Dentro desse contexto, o processamento de produtos agropecuários para autoconsumo é uma prática tradicional entre os agricultores familiares, destacando-se como uma atividade significativa para a subsistência e a economia local. No Brasil, especificamente, o setor agrícola exerce função de destaque na economia, contribuindo com 7,1% do Produto Interno Bruto (PIB) nacional, gerando empregos e garantindo a segurança alimentar não apenas do país, mas também de diversas nações ao redor do mundo (GOV..., 2024). Esses dados ressaltam a importância dos pequenos agricultores e suas práticas na sustentabilidade alimentar e econômica global. Entretanto, apesar do papel fundamental desempenhado por esses agricultores, ainda há desafios a serem superados para maximizar seu potencial produtivo. Um desses desafios é a adoção de tecnologias, como o uso de software, que têm sido um grande aliado do agronegócio nas últimas décadas.

A falta de adoção de Sistemas de Informação (SI) nas propriedades rurais pode ser influenciada por diversos fatores, como, a carência de informação e capacitação dos agricultores, a escassez de incentivo e subsídios governamentais e a falta de adequação do mercado às necessidades reais desse segmento. Além disso, é importante ressaltar que diversos outros fatores podem influenciar na baixa adoção de SI na agricultura de subsistência familiar. Entre esses fatores, destacam-se a resistência por parte dos minifundiários em se adaptar ao uso de novas tecnologias e a prioridade de manter uma gestão financeira equilibrada. Esses aspectos, muitas

vezes, levam os agricultores a negligenciarem investimentos em SI, por não perceberem o valor dessas soluções para o gerenciamento de suas atividades agrícolas.

Considerando a importância do gerenciamento das operações em uma propriedade rural e a busca por maximizar a eficiência e reduzir os custos de produção, é importante que os pequenos agricultores reconheçam a importância em usar um SI para gerenciar seu negócio e simplificar suas atividades diárias.

Diante desse contexto, foi desenvolvido um sistema *web* para atender às demandas de gestão de uma pequena propriedade rural. O objetivo principal é proporcionar um maior controle financeiro e reduzir o tempo e esforço dedicado à realização de registros manuais em agendas e calendários físicos. Para alcançar esse propósito, a aplicação conta com recursos como o cadastro de categorias para controle de movimentações financeiras, gestão de estoque de insumos como ração e ativos imobilizados, além de funcionalidades para previsão de natalidade do gado.

1.2 Objetivos

A seguir são apresentados os objetivos do sistema proposto neste trabalho, sendo o objetivo geral o resultado principal esperado e os objetivos específicos as principais funcionalidades do sistema.

1.2.1 Objetivo Geral

Desenvolver um sistema *web* para gestão financeira e operacional de pequenas propriedades rurais.

1.2.2 Objetivos Específicos

- Possibilitar o cadastro de categorias para controle de entradas e saídas de caixa, permitindo a gestão financeira da propriedade.
- Emitir mensagens de alerta sobre vencimentos de contas a pagar.
- Possibilitar o controle de estoque de rações e insumos, incluindo registro de uso e venda.

- Possibilitar classificação de itens do estoque e emissão de relatórios detalhados de entradas, saídas e saldo.
- Visualizar e prever a taxa de natalidade de bovinos, possibilitando a geração de relatórios detalhados que incluam informações como a data de concepção ou de inseminação (em casos de reprodução artificial).

1.3 Justificativa

A agricultura familiar contribui para a economia global, fornecendo alimentos para grande parte da população e contribuindo significativamente para o desenvolvimento socioeconômico das áreas rurais. No entanto, os pequenos agricultores enfrentam desafios específicos, incluindo a falta de acesso a tecnologias e ferramentas de gestão eficientes que possam otimizar suas operações e garantir a sustentabilidade de suas atividades.

Diante desse contexto, a proposta de desenvolvimento de um sistema *web* para gestão de pequenas propriedades rurais surge como uma iniciativa para superar as barreiras enfrentadas pelos agricultores familiares. Ainda que o uso de softwares tenha se popularizado no agronegócio, observa-se uma lacuna na adoção dessas tecnologias por pequenas propriedades, principalmente devido à resistência dos produtores em se adaptar a novas ferramentas e à necessidade de manter as finanças equilibradas.

Nesse sentido, o desenvolvimento desse trabalho se justifica pela iniciativa em propor uma solução tecnológica acessível e adaptada às necessidades específicas dos pequenos agricultores. Por meio do desenvolvimento de um sistema de gestão financeira, controle de estoque e previsão de natalidade do gado, pretende-se fornecer aos agricultores familiares uma solução que seja útil para auxiliar na gestão visando reduzir custos e aumentar a produtividade de suas propriedades. O sistema também visa simplificar o processo de gestão, substituindo métodos tradicionais, como anotações em agendas e calendários físicos, por uma plataforma digital intuitiva e de fácil utilização. Espera-se, assim, que a implementação dessa solução contribua não apenas para o aumento da eficiência e competitividade das pequenas propriedades rurais, mas também para a melhoria da qualidade de vida dos agricultores familiares e o fortalecimento da agricultura familiar como um todo.

Além disso, a implementação desse sistema poderá incentivar os agricultores familiares na adoção de tecnologias, o que pode resultar em uma maior conectividade com mercados, acesso a informações sobre preços e tendências de mercado, e uma melhor capacidade de planejamento e tomada de decisões.

Conforme destacado pela Embrapa em 2011, os sistemas de informação gerenciais destinados às propriedades rurais auxiliam na viabilidade e prosperidade dos empreendimentos agrícolas (EMBRAPA, 2011). Essas ferramentas têm a capacidade de oferecer suporte às atividades administrativas, bem como facilitar o acompanhamento e a gestão das atividades produtivas. A adoção de SI nas pequenas propriedades rurais pode contribuir para a inclusão digital no meio rural, incluir melhorias na organização e controle das operações, reduzir custos e o fortalecer a competitividade do negócio agrícola.

2 REFERENCIAL TEÓRICO

Este capítulo aborda conceitos fundamentais para a compreensão da agricultura familiar, a gestão de propriedades rurais e a aplicação de tecnologias no agronegócio. Serão discutidos os benefícios da adoção dessas tecnologias, tanto para grandes quanto para pequenas propriedades rurais, destacando as vantagens econômicas, ambientais e operacionais.

2.1 Agricultura familiar

A agricultura familiar tem desempenhado um papel importante no desenvolvimento econômico, social e ambiental de diversas regiões, principalmente em países em desenvolvimento. Este modelo agrícola é caracterizado por propriedades de pequeno porte, onde o trabalho familiar é o principal motor da produção, e seus produtos frequentemente abastecem mercados locais e regionais. A agricultura familiar é importante para promover a sustentabilidade das comunidades rurais e contribuir significativamente para a manutenção da biodiversidade, a conservação dos recursos naturais e a promoção de uma alimentação saudável (Knob, 2016).

A agricultura familiar se distingue da agricultura empresarial por seu foco na produção voltada para o consumo local e pela menor dependência de insumos externos, o que reduz o impacto ambiental da produção. Além disso, estudos como o de Altieri (2012) enfatizam o papel da agricultura familiar na preservação de variedades locais de culturas e na manutenção dos saberes tradicionais relacionados ao cultivo da terra, que são passados de geração em geração.

Em contrapartida, a agricultura familiar enfrenta desafios, como o acesso limitado a crédito, políticas públicas inadequadas e a pressão da agricultura industrial. Contudo, Knob (2016) argumenta que a valorização da agricultura familiar, por meio de políticas públicas mais inclusivas e sustentáveis, pode potencializar seu impacto positivo tanto na economia quanto na preservação ambiental. A adoção de tecnologias sustentáveis, como a agroecologia, tem sido uma das alternativas apontadas por pesquisadores como outra forma de garantir a viabilidade e a permanência da agricultura familiar (Teixeira *et al*., 2010).

Portanto, é imprescindível que haja uma integração entre os saberes tradicionais e as novas tecnologias agrícolas, além do apoio governamental, para fortalecer a agricultura familiar. Além disso, a gestão eficiente das propriedades

agrícolas é essencial para o sucesso e a sustentabilidade das atividades no campo. A crescente demanda por alimentos e a necessidade de maximizar a produtividade têm levado à adoção de novas tecnologias e ferramentas que auxiliam no gerenciamento das operações agrícolas.

2.2 Gestão de propriedades agrícolas

Os SI têm se mostrado importantes aliados para os agricultores, proporcionando melhor controle sobre as diversas variáveis que influenciam a produção, como clima, solo, uso de insumos e logística. A utilização de SI na gestão agrícola permite o monitoramento das condições da propriedade, otimizando o uso de recursos e melhorando a tomada de decisões. Segundo Lemos *et al.* (2017), os SI integrados oferecem soluções para o gerenciamento de dados de forma eficiente, desde o planejamento até a execução da produção, permitindo uma visão mais detalhada e precisa do ciclo produtivo. Estes sistemas, ao facilitar o acesso a informações, tornam os processos mais ágeis, ajudam na automação de tarefas e garantem o cumprimento de regulamentos ambientais e de qualidade.

Além disso, o uso de tecnologias de informação pode contribuir significativamente para o desenvolvimento da agricultura de precisão, que se baseia na coleta e análise de dados para otimizar a produtividade e minimizar desperdícios. A agricultura de precisão, como descrito por Nogueira *et al.* (2020), envolve a utilização de tecnologias como sensores, *Global Positioning System* (GPS) e análise de dados para monitorar e gerenciar as variáveis ambientais e produtivas de uma propriedade, resultando em decisões mais acertadas e eficientes. Pierce e Nowak (1999) destacam ainda que a integração tecnológica no campo promove uma gestão mais eficiente dos recursos naturais, tornando o setor mais sustentável e competitivo. De acordo com Knob (2016), a utilização de tecnologias de precisão em propriedades de menor escala pode ser adaptada para promover uma agricultura mais eficiente e ambientalmente responsável.

No caso das pequenas propriedades rurais, a implementação dessas tecnologias pode ser ainda mais desafiadora devido às limitações financeiras e de infraestrutura. Contudo, ao adotar ferramentas mais acessíveis e adaptadas à realidade dessas propriedades, é possível alcançar ganhos significativos em termos de eficiência e sustentabilidade. A agricultura de precisão pode ser aplicada em menor escala, desde que haja uma análise detalhada das necessidades e das condições

específicas de cada propriedade, permitindo a otimização de insumos, a melhoria na gestão de recursos hídricos e a maximização do uso da terra. Segundo Silva *et al.* (2019), a tecnologia permite que pequenos produtores possam aumentar a produtividade sem expandir as áreas cultivadas, utilizando de maneira mais eficaz os recursos disponíveis. Além disso, sistemas de gestão simplificados podem ser adaptados para essas propriedades, oferecendo maior controle sobre as operações e auxiliando na tomada de decisões estratégicas.

No entanto, a implementação de SI na gestão agrícola ainda enfrenta desafios, como a falta de infraestrutura tecnológica em algumas regiões rurais, a necessidade de capacitação dos agricultores e a adaptação dos sistemas às especificidades de cada propriedade. Esses obstáculos podem ser superados por meio de políticas públicas que incentivem a adoção de tecnologias e promovam a educação digital no meio rural.

A integração da agricultura de precisão com sistemas de gestão representa uma oportunidade estratégica para enfrentar os desafios do setor, proporcionando benefícios econômicos, ambientais e sociais. Assim, o desenvolvimento de ferramentas tecnológicas personalizadas para atender às especificidades das propriedades rurais é fundamental para a evolução do agronegócio.

2.3 Automação no agronegócio

O agronegócio se refere a todas as atividades econômicas relacionadas ao comércio de produtos agrícolas, que abrange os mais variados perfis, subdivido em três grandes setores, também chamados de níveis (FIA, 2021):

- Setor Primário: Inclui os produtores rurais, agricultores e pecuaristas, responsáveis pela produção de matéria-prima agrícola e pecuária.
- Setor Secundário: Envolve as agroindústrias e fabricantes de insumos, que processam as matérias-primas e produzem bens intermediários, como alimentos industrializados, máquinas agrícolas, fertilizantes e defensivos.
- Setor Terciário: Engloba transportadoras, distribuidores e comerciantes de produtos agrícolas, responsáveis pela logística, distribuição e comercialização dos produtos para o consumidor final.

Como destaca a equipe TOTVS (2024), o objetivo primário da agricultura comercial é gerar lucro, o que a distingue da agricultura de subsistência. A busca pelo lucro impulsiona a adoção de tecnologias mais avançadas, a otimização da produção e a especialização em cultivos rentáveis, enquanto a agricultura de subsistência prioriza a autossuficiência e a diversidade de cultivos. Tendo isso em vista, pode-se observar que os softwares para grandes latifúndios possuem, em sua grande maioria, ferramentas para emissão de Notas Fiscais Eletrônicas (NF-e), previsão de rentabilidade da lavoura e outras atividades fiscais e financeiras, como controle de transporte e de folha de pagamentos de funcionários.

2.4 Trabalhos relacionados

Existe uma grande gama de softwares para gestão para cada setor do agronegócio, com especialidades únicas e personalizações para atender a demanda em sua totalidade. Por exemplo, um software para o setor primário é focado no comércio e distribuição de produtos, o que inclui: gestão de transporte e logística, gestão de relacionamento com clientes, controle de vendas, estoque e faturamento, suporte a e-commerce e integrações robustas.

A seguir, são descritos brevemente alguns softwares que visam a gestão de propriedades rurais. Todos eles possuem a funcionalidade de emissão de NF-e em comum:

Perfarm: é um sistema *web* voltado para a gestão de pequenas e médias propriedades rurais, auxiliando no controle de atividades agrícolas, pecuárias e financeiras. Ele integra ferramentas para planejamento, monitoramento e análise de dados, otimizando a produtividade e a tomada de decisões e a visualização da previsão de saldo futuro de contas bancárias e de estoque.

Aegro: é um sistema *web* e aplicativo móvel voltado para gestão agrícola, oferecendo ferramentas para controle financeiro, operacional e técnico. Conta com planejamento de safra, imagens de satélite e conexão com máquinas.

Agrotitan: é um software para *desktop* que oferece solução para a indústria agropecuária, oferecendo uma variedade de recursos, incluindo absorção de custos, fluxo de produção, controle de perdas e rastreabilidade da produção. Também possui ferramentas para gestão de Recursos Humanos (RH).

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizado para o desenvolvimento do sistema.

3.1 Materiais

O Quadro 1 apresenta as principais ferramentas e tecnologias utilizadas no desenvolvimento deste projeto.

Quadro 1 - Lista de Ferramentas e Tecnologias

Ferramenta / Tecnologia	**Versão**	**Finalidade**
Java	21.0.2	Linguagem de Programação *back-end*
React JS	18.2	*Framework front-end*
TypeScript	5.8	Linguagem de Programação *front-end*
HTML	5	Linguagem de marcação
CSS	3	Linguagem de estilo
Tailwind CSS	4.1.17	*Framework* utilitário para estilização do front-end
Shadcn/ui	3.5.0	Biblioteca de componentes estilizados com Tailwind
PostgresSQL	16.2	Banco de dados
Springboot	3.2.2003	*Framework back-end*
IntelliJ IDEA	Ultimate	Ambiente de desenvolvimento
Visual Studio Code	1.87	Ambiente de desenvolvimento
Hibernate	6.4	*Framework* para mapeamento objeto-relacional
Recharts	3.3.0	Biblioteca para criação de gráficos em React
Whapi.cloud	1.8.7	API para envio de notificações via WhatsApp
Draw.io	29.0.3	Ferramenta para diagramas de caso de uso

Fonte: Autoria própria (2025).

A seguir serão descritas as principais tecnologias utilizadas no desenvolvimento do trabalho.

3.1.1 REACT

React é uma biblioteca JavaScript voltada para a construção de interfaces de usuário (UI). Assim como outras bibliotecas, como Vue e Angular, o React se destaca por sua abordagem eficiente e flexível. Lançado pelo Facebook em 2013, o React se tornou amplamente popular após o código ser disponibilizado como *open source*. Essa popularização é refletida no grande número de downloads registrados em 2019 e 2020. Muitas grandes empresas, como American Express, Netflix e Airbnb, adotaram o React, o que impulsionou ainda mais seu sucesso. As constantes atualizações do Facebook, criador da biblioteca, têm sido fundamentais para manter o React como uma das principais ferramentas para desenvolvimento de interfaces.

De acordo com a documentação oficial, React é descrito como uma biblioteca JavaScript " declarativa, eficiente e flexível para criar interfaces de usuário. Ele permite compor interfaces complexas a partir de pequenos e isolados códigos chamados componentes" (META, 2019). Essa abordagem declarativa traz diversas vantagens sobre a programação imperativa, onde o código detalha precisamente os passos necessários para realizar uma tarefa. No caso do React, a linguagem declarativa permite que os desenvolvedores se concentrem no que a aplicação deve fazer, sem a necessidade de definir como cada tarefa será realizada, economizando tempo e esforço.

Entre as características mais importantes do React estão os Componentes, que são blocos reutilizáveis de código responsáveis pela construção da interface de usuário. Cada componente pode gerenciar seu próprio estado e renderizar dados, tornando a manutenção e o desenvolvimento mais eficientes. A utilização do JSX (JavaScript XML) facilita a combinação de HTML e JavaScript, permitindo a criação de componentes com uma sintaxe mais próxima ao HTML, mas mantendo a flexibilidade e o poder do JavaScript. Além disso, o conceito de *Document Object Model* (DOM) e Virtual DOM é essencial para entender como o React gerencia a renderização da interface.

O DOM é uma representação em árvore dos elementos de uma página *web*. Cada elemento HTML na página é um nó dentro dessa árvore, e o navegador usa o DOM para manipular e renderizar a interface do usuário. Porém, quando há mudanças no DOM, o navegador precisa recalcular a renderização, o que pode ser um processo demorado e ineficiente, principalmente em interfaces complexas.

Por outro lado, o Virtual DOM é uma versão otimizada do DOM utilizada pelo React. Quando ocorre uma alteração no estado de um componente, o React primeiro atualiza o Virtual DOM e, em seguida, compara essa nova versão com a versão anterior (esse processo é conhecido como "reconciliação"). Somente as partes do DOM que realmente sofreram alterações são atualizadas no navegador. Isso torna a renderização mais eficiente, pois o React minimiza o número de operações no DOM real, reduzindo o impacto no desempenho da aplicação.

Outro conceito central do React é o fluxo de dados unidirecional. Isso significa que os dados fluem de um ponto central (geralmente o estado do componente) para as partes da aplicação que dependem deles. Esse modelo facilita a previsibilidade do comportamento da aplicação, tornando a depuração e a manutenção mais simples. Além disso, a introdução dos *Hooks* no React 16.8 trouxe uma maneira de usar o estado e outros recursos do React em componentes funcionais, o que simplifica o código e melhora a legibilidade.

A principal vantagem do React está em sua capacidade de reutilização de componentes. Isso permite que desenvolvedores criem interfaces modulares e escaláveis, onde os componentes podem ser reutilizados em diferentes partes da aplicação. Isso não só melhora a produtividade, mas também reduz o risco de erros, já que componentes reutilizáveis são testados e validados individualmente. Outro benefício significativo do React é o seu desempenho, que é otimizado por meio do uso do Virtual DOM e da atualização eficiente da interface de usuário. Essas características tornam o React uma escolha sólida para aplicações que exigem alta interatividade e performance.

Portanto, React não é apenas uma biblioteca para criar interfaces, mas uma ferramenta poderosa que permite aos desenvolvedores construir aplicações complexas de forma mais eficiente, modular e escalável.

3.1.2 Linguagem Java

A linguagem Java é amplamente utilizada em aplicações corporativas devido à sua robustez, portabilidade e segurança, incluindo sistemas *desktop*, *web* e aplicações móveis. Além disso, possui uma vasta biblioteca de *Application Programming Interface* (API) e uma comunidade ativa, o que facilita a resolução de problemas e a implementação de soluções de forma eficiente.

Criada pela Sun Microsystems em 1995 e posteriormente adquirida pela Oracle Corporation, Java é conhecida por ser uma linguagem robusta, segura e portátil, devido ao uso da *Java Virtual Machine* (JVM) (Deitel; Deitel, 2015).

Uma das principais características do Java é sua orientação a objetos, o que permite organizar o código em torno de conceitos do mundo real, como classes e objetos, promovendo a reutilização e a manutenção de código. Além disso, a linguagem é projetada para ser independente de plataforma (Oracle, 2025).

Sua popularidade e suporte a diferentes plataformas tornam a linguagem Java uma escolha confiável para sistemas complexos e de larga escala. Por ser uma linguagem orientada a objetos, permite um *design* de software mais organizado e reutilizável, favorecendo a manutenção e escalabilidade do sistema.

Java possui uma vasta biblioteca padrão (Java API), que facilita o desenvolvimento de funcionalidades como manipulação de arquivos, redes, gráficos e acesso a banco de dados.

Com o advento de *frameworks* como Spring e Hibernate, Java consolidou-se como uma das principais escolhas para o desenvolvimento de sistemas corporativos e *web* (Gosling *et al*., 2000). Entre as edições de Java disponíveis, destacam-se:

- Java *Standard Edition* (Java SE): Para aplicações *desktop* e básicas.
- Java *Enterprise Edition* (Java EE): Para aplicações corporativas de grande porte.
- Java *Micro Edition* (Java ME): Para dispositivos móveis e embarcados.

Além disso, Java continua evoluindo com versões regulares que trazem novos recursos e melhorias. Algumas das versões mais recentes introduziram recursos como expressões lambda, API de fluxos (*Streams* API) e, mais recentemente, a funcionalidade de *Records*, que facilita a criação de classes imutáveis.

3.1.3 Spring Framework

O Spring é um ecossistema de projetos voltado para melhorar a produtividade dos desenvolvedores, oferecendo simplicidade e flexibilidade no desenvolvimento de aplicações Java. Entre os *frameworks* do Spring, destacam-se o Spring MVC (*Model-View-Controller*), para criação de aplicações *web* e serviços *Representational State Transfer* (REST); o Spring Data, que facilita o acesso a bancos de dados; e o Spring Security, que fornece soluções de segurança para aplicações. Além disso, o

ecossistema abrange diversos outros projetos voltados para computação em nuvem, e microsserviços, por exemplo.

O Spring MVC é um dos módulos mais utilizados do Spring *framework*, amplamente reconhecido por facilitar o desenvolvimento de aplicações *web* robustas e escaláveis. Baseado no padrão arquitetural MVC, esse módulo organiza o sistema em camadas distintas, reduzindo o acoplamento entre elas e promovendo maior flexibilidade e manutenção do código.

O padrão MVC foi idealizado em 1979 pelo cientista norueguês Trygve Reenskaug, enquanto trabalhava na Xerox PARC, com o objetivo de estruturar sistemas de forma modular e organizada. Apesar de sua criação ter ocorrido há décadas, a popularidade do MVC cresceu significativamente com a evolução do desenvolvimento *web*, sendo amplamente adotado por suas vantagens em termos de clareza e manutenção do software (Silva, 2020).

De acordo com Silva (2020), o padrão MVC divide a aplicação em três camadas principais:

- *Model*: responsável pela manipulação e gerenciamento dos dados.
- *View*: encarregada da interface com o usuário.
- *Controller*: atua como intermediário, processando as requisições do usuário e coordenando a interação entre *Model* e *View*.

No contexto de uma aplicação *web*, ao receber uma solicitação, o *Controller* processa a requisição, comunica-se com o *Model* para obter os dados necessários e os repassa para a *View*, que então apresenta o resultado ao usuário.

Essa organização em camadas permite que o desenvolvimento seja mais modular e facilita a manutenção, características que justificam a ampla adoção do padrão MVC em *frameworks* modernos como o Spring MVC.

Segundo a documentação oficial (Spring, 2024), na versão 2.5 foi introduzido um modelo de programação baseado em anotações para controladores MVC que usa anotações como *@RequestMapping*, usado para mapear uma requisição *Hipertext Transfer Protocol* (HTTP), *@RequestParam*, usado para acessar parâmetros de requisição passados na *Uniform Resource Locator* (URL) ou no corpo da requisição (como parâmetros de formulário) e o *@ModelAttribute*, utilizado para vincular parâmetros de uma requisição a um objeto do modelo ou para adicionar atributos ao modelo antes da execução do método do controlador e assim por diante. A documentação também destaca que os controladores implementados nesse estilo não

precisam estender classes base específicas ou implementar interfaces específicas (*Spring*, 2024).

Para atuar na camada de persistência (Spring, 2024), foi utilizado o *framework* Spring Data JPA (*Java Persistence API*), para trabalhar com bancos de dados relacionais usando a especificação JPA. Ele auxilia o programador na criação dos repositórios da aplicação. O projeto Spring Data JPA faz parte do Spring Data, que inclui diversos outros projetos para facilitar o acesso e a persistência de dados.
Entre as principais funcionalidades do Spring Data, destacam-se (Spring, 2024):

- Abstrações para repositórios e mapeamento de objetos personalizados.
- Derivação dinâmica de consultas a partir de nomes de métodos nos repositórios.
- Classes base de domínio com propriedades comuns predefinidas.
- Suporte para auditoria automática, como registro de criação e última modificação.
- Integração de código personalizado em repositórios.
- Configuração simplificada com suporte a JavaConfig e *Extensible Markup Language* (XML).
- Integração avançada com controladores do Spring MVC.

Por meio da utilização dessa ferramenta, o desenvolvedor pode apenas definir as interfaces dos repositórios, e o Spring se encarrega de implementar automaticamente as operações. Além disso, o Spring Data JPA suporta técnicas avançadas, como "*query by example*" e consultas (*queries*) personalizadas, escrevendo as *queries* necessárias automaticamente (*Spring*, 2024).

A segurança da aplicação foi gerenciada utilizando o Spring Security, um módulo do Spring Framework projetado para oferecer autenticação, autorização e outras medidas de proteção de forma robusta e personalizável. De acordo com a documentação oficial (Spring, 2024), o *framework* suporta uma ampla gama de cenários, desde autenticação baseada em formulários até integrações com sistemas de identidade como *Open Authorization* 2.0 (*OAuth2*) e *Security Assertion Markup Language* (SAML).

O Spring Security segue o princípio de "segurança por padrão", fornecendo configurações iniciais que podem ser adaptadas conforme as necessidades específicas da aplicação. Ele se integra facilmente com outros módulos do Spring,

como Spring MVC, permitindo a implementação de regras de acesso baseadas em URLs e funções de usuários, além de recursos avançados, como proteção contra ataques *Cross-Site Request Forgery* (CSRF) e configuração de cabeçalhos HTTP para segurança adicional.

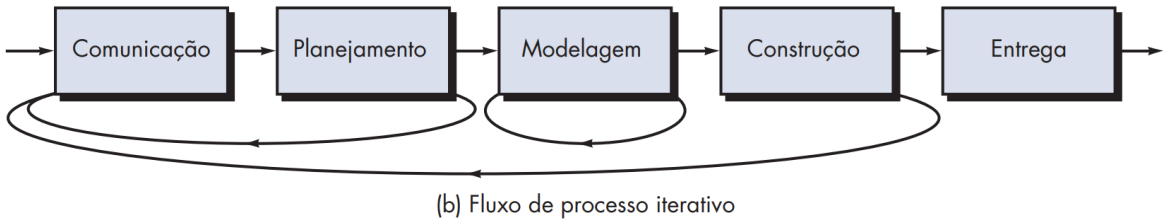
Conforme descrito por Garcia (2021), o Spring Security também oferece suporte a métodos declarativos e programáticos para controle de acesso. Isso possibilita aplicar segurança tanto em nível de API quanto diretamente no código, por meio de anotações como *@PreAuthorize* e *@PostAuthorize*. Além disso, sua estrutura modular facilita a configuração de autenticação utilizando bancos de dados, serviços *Lightweight Directory Access Protocol* (LDAP), ou provedores externos, como Google e Facebook.

Essa flexibilidade torna o Spring Security uma escolha ideal para garantir que a aplicação atenda aos mais altos padrões de segurança, oferecendo uma base sólida para proteger dados sensíveis e preservar a integridade dos sistemas.

3.2 Método

Para o desenvolvimento do projeto, foi adotada a metodologia incremental proposta por Pressman e Maxim (2021), composta por cinco atividades principais: comunicação, planejamento, modelagem, construção e entrega. Todas essas etapas ocorrem de maneira contínua e iterativa até o fim do projeto, isso é, as atividades são realizadas de forma constante e repetitiva, com ciclos de aprimoramento e adaptação ao longo do desenvolvimento do projeto, visando garantir um controle eficaz do progresso, gerenciamento de riscos, qualidade do produto, administração de configurações e realização de revisões técnicas periódicas. Esse modelo de processo pode ser visualizado na Figura 1.

Figura 1- Fluxo de processo iterativo



(b) Fluxo de processo iterativo

Fonte: Pressman e Maxim (2021, p. 83)

O fluxo iterativo, apresentado na Figura 1, permite que as atividades sejam repetidas antes de avançar para a próxima fase, desenvolvendo o sistema de maneira progressiva e em incrementos funcionais. Cada ciclo revisita e aprimora aspectos do produto, realizando ajustes e melhorias contínuas com base em *feedback* dos clientes e/ ou da equipe de desenvolvimento, garantindo um processo de desenvolvimento flexível e adaptável. A seguir será descrita, de maneira detalhada, cada uma das atividades do modelo utilizado.

Comunicação: essa atividade tem como objetivo reunir requisitos que ajudem na definição dos recursos e funções do sistema, por meio da delimitação de seu escopo. O projeto busca atender à demanda de gerenciamento de propriedades familiares surgindo a partir da observação de um processo manual de preenchimento de agenda em papel, utilizado como ferramenta organizacional. Para aprofundar o entendimento sobre as dificuldades e limitações desse método tradicional, foram realizadas conversas com os principais responsáveis pela gestão, identificando desafios enfrentados no cotidiano. O sistema proposto visa solucionar problemas, proporcionando maior praticidade e eficiência na consulta e no gerenciamento das informações registradas. Além disso foi realizada uma pesquisa sobre sistemas semelhantes existentes no mercado, conforme descrito no Capítulo 2.

Planejamento: essa fase envolve a descrição das tarefas técnicas a serem desenvolvidas, a identificação dos recursos necessários e elaboração de um cronograma de trabalho, incluindo os objetivos de cada iteração. Embora o planejamento inicial seja estabelecido no início do projeto, ele é continuamente ajustado ao longo do desenvolvimento, conforme o progresso do projeto e o refinamento do software. Durante a etapa de Proposta de Trabalho de Conclusão de Curso 1 (TCC1), foram definidos os prazos de entregas de algumas etapas, o desenho do escopo e o levantamento dos objetivos do projeto.

Modelagem: nessa etapa foram criadas representações do sistema, como os diagramas de entidade-relacionamento, diagramas de casos de uso e tabelas de requisitos funcionais e não funcionais. O objetivo dessa atividade é capturar e estruturar os requisitos do sistema, representando suas entidades, funcionalidades e características de forma clara e compreensível. Por ser realizada de maneira incremental, a modelagem permite que, cada nova iteração, todos esses elementos sejam revisitados e ajustados, promovendo melhorias contínuas.

Construção: essa etapa abrange o conjunto de atividades relacionadas à de codificação e testes. A codificação envolve a criação das interfaces e o desenvolvimento do código-fonte, implementando as funcionalidades de acordo com os requisitos estabelecidos. Os testes serão realizados conforme a finalização de cada tarefa e/ou tela.

Entrega: a entrega final do projeto apresentado no Trabalho de Conclusão de Curso 2 (TCC2), contém a documentação das interfaces e códigos-fonte. O sistema está configurado para uso em um ambiente acadêmico experimental, permitindo sua validação em situações práticas. Havendo interesse, o projeto poderá ser ampliado para incorporar novas funcionalidades e atingir um público maior.

4 RESULTADOS

Este capítulo apresenta os resultados deste trabalho, que consiste no desenvolvimento de um sistema *web* para gestão de pequenas propriedades rurais. O capítulo, apresenta o escopo do sistema, a modelagem, que envolve os requisitos funcionais e não-funcionais, diagrama de casos de uso e do banco de dados.

4.1 Escopo

O sistema proposto neste trabalho visa oferecer uma solução para a gestão administrativa e operacional de propriedades rurais, garantindo controle das atividades diárias. Primeiramente, é importante ressaltar que o proprietário, ou aquele responsável pela gestão da propriedade, também conhecido como “caseiro”, será o principal usuário do sistema, dado que propriedades desse porte geralmente não contam com funcionários ou possuem apenas uma equipe reduzida, então a alimentação do sistema completo tende a ser realizada pelo mesmo usuário.

O sistema permite o gerenciamento das finanças da propriedade, incluindo o controle de caixa, contas bancárias e saldos em carteira física, além do acompanhamento das contas a receber e pagar por meio do cadastro de categorias/plano de contas. Também disponibiliza a inclusão e edição de parcelas retroativas e lançamentos futuros para assegurar previsibilidade financeira. Além disso, permite emitir relatórios financeiros para visualização das movimentações e resultados do período selecionado, com a possibilidade de incorporar gráficos e *dashboards* na página inicial para uma análise mais detalhada. Para melhorar a comunicação com o público-alvo, foram desenvolvidos gatilhos de notificação por WhatsApp, para alertar sobre vencimentos de contas a pagar e fornecer orientações sobre os pagamentos, considerando que parte do público pode ter menor familiaridade com tecnologia. Outra funcionalidade importante é o controle de estoque de rações e insumos, permitindo informar número exato de produtos disponíveis, bem como o registro de sua utilização e venda. É possível classificar os itens de acordo com sua finalidade, como consumo, comercialização, devolução ou empréstimo. Além disso, o sistema contempla a emissão de relatórios detalhados de entradas, saídas e saldo de cada item no estoque. O sistema também possui o controle de rebanhos, incluindo

informações como vacinas aplicadas, peso e raça, além da previsão de data de parto para as fêmeas, visando garantir a gestão da criação e previsibilidade da natalidade.

Para uma gestão comercial mais completa, oferece ainda a possibilidade de incluir clientes e fornecedores, que serão os atores externos, mantendo um registro organizado das relações comerciais. O sistema conta com telas totalmente responsivas e, em versões futuras, poderá contar com uma versão *mobile* para maior acessibilidade e praticidade de uso em diferentes dispositivos.

4.2 Modelagem do Sistema

A seguir são apresentados os requisitos funcionais e não funcionais do sistema, os casos de uso e o diagrama do banco de dados. Esses elementos fornecem uma visão abrangente do projeto, contemplando as funcionalidades previstas, as restrições e as necessidades técnicas, além de demonstrar como os dados serão estruturados e administrados.

4.2.1 Requisitos funcionais e não funcionais

Os requisitos funcionais representam os serviços e as funcionalidades que o software disponibiliza ao usuário, definidos com o objetivo de atender às suas necessidades. Já os requisitos não funcionais, conforme descrito por Chung (2009), correspondem a aspectos que não se relacionam diretamente com as funções do sistema, mas que influenciam sua qualidade, como usabilidade, segurança, clareza, escalabilidade, entre outras características essenciais.

O Quadro 2 apresenta os requisitos funcionais do sistema, contemplando todas as funcionalidades para a gestão agrícola.

Quadro 2 - Requisitos Funcionais

Identificação	Nome	Descrição
RF01	Manter usuários	O cadastro de usuários do sistema deve conter as informações básicas, como nome, telefone de contato, e-mail e senha. Essas informações são essenciais para garantir a identificação e autenticação dos usuários no sistema.
RF02	Manter finanças	O sistema deve permitir o gerenciamento das finanças da propriedade, incluindo controle de caixa, contas bancárias e saldos em carteira física.
RF03	Manter estoque	O sistema deve permitir o controle, a inclusão, a edição e o gerenciamento de estoque de rações e insumos, registrando a utilização e venda dos produtos.

RF04	Manter rebanho de gado	O sistema deve realizar o gerenciamento de entrada e saída de rebanhos, incluindo informações como vacinas aplicadas, peso, raça e previsão de parto.
RF05	Manter clientes e fornecedores	O sistema deve permitir a inclusão, a edição e o gerenciamento de clientes e fornecedores, mantendo registros organizados das relações comerciais.
RF06	Manter agenda e atividades	O sistema deve permitir o agendamento e controle de atividades planejadas, como vacinação, manutenção de equipamentos e tarefas gerais, enviando lembretes automáticos ao usuário por e-mail ou WhatsApp.
RF07	Gerar relatórios	O usuário poderá acompanhar o desempenho do estabelecimento por meio de *dashboards* e relatórios específicos de vendas, estoque e financeiro.
RF08	Configurar notificações	O sistema deve enviar notificações por e-mail ou WhatsApp para alertar sobre vencimentos de contas a pagar e fornecer orientações sobre pagamentos.

Fonte: Autoria própria (2025).

O Quadro 3 apresenta os requisitos não funcionais do sistema, descrevendo as características técnicas e de qualidade que garantirão o bom funcionamento, segurança e usabilidade da aplicação.

Quadro 3 - Requisitos Não Funcionais

Identificação	Nome	Descrição
RNF01	Segurança de dados	O sistema deve garantir a proteção dos dados dos usuários por meio de autenticação segura, criptografia de senhas e uso de HTTPS para comunicações.
RNF02	Performance	O sistema deve ser capaz de processar operações de cadastro, edição e relatórios em até 3 segundos para até 100 usuários simultâneos.
RNF03	Usabilidade	A interface do sistema deve ser intuitiva, responsiva e acessível, com design adaptado para uso em dispositivos móveis e computadores.
RNF04	Escalabilidade	O sistema deve permitir fácil expansão para inclusão de novas funcionalidades e integração com APIs externas, como softwares de contabilidade ou dispositivos de rastreio.
RNF05	Portabilidade	O sistema deve ser compatível com os navegadores modernos (Chrome, Firefox, Edge) e operar em sistemas Windows, macOS, iOS e Android.
RNF06	Disponibilidade	O sistema deve estar disponível em tempo integral, exceto durante períodos de manutenção programada, que devem ser informados com antecedência ao usuário.

RNF07	*Backup* e recuperação	O sistema deve realizar backups automáticos diários dos dados e oferecer um mecanismo para restaurar dados em caso de falhas ou exclusões acidentais.
RNF08	Multiplataforma	O sistema deve ser desenvolvido como uma aplicação *web* responsiva, mas com possibilidade de evoluir para uma aplicação *mobile* nativa em versões futuras.
RNF09	Integração com notificações	O sistema deve integrar-se com serviços de terceiros (como APIs do WhatsApp e *Simple Mail Transfer Protocol* (SMTP) para e-mail) para envio de notificações automatizadas.

Fonte: Autoria própria (2025).

4.2.2 Casos de Uso

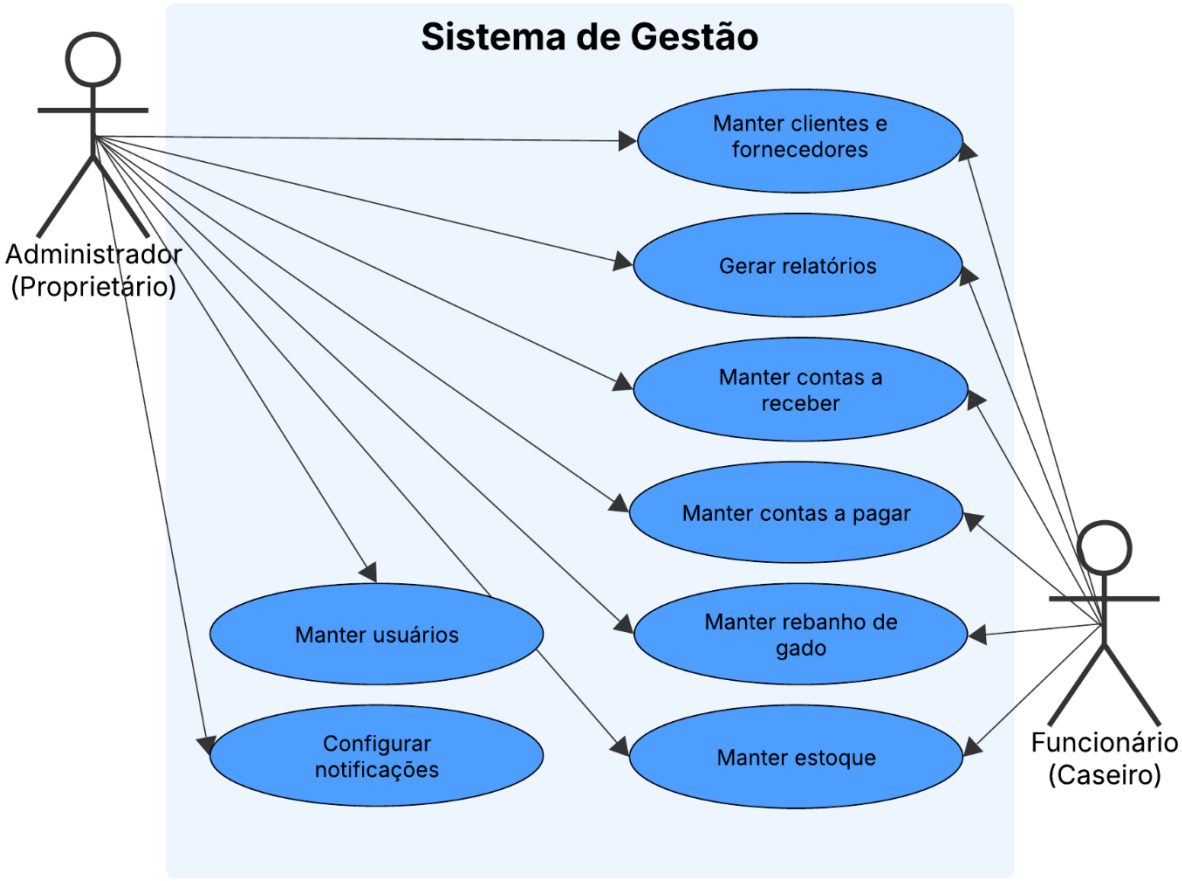
A Figura 2 apresenta o diagrama de Casos de Uso do sistema. Esse diagrama é uma das formas de demonstrar, por meio da Linguagem de Modelagem Unificada (UML), as interações entre os usuários e a aplicação. Nele são representados os atores, que são as entidades responsáveis por executar ações dentro do sistema, assim como as funcionalidades disponíveis.

O sistema possui dois tipos de usuários: o Administrador, que geralmente é o proprietário da propriedade, e o Funcionário, responsável por atividades operacionais, também conhecido como “Caseiro”.

Além dos atores humanos, o próprio sistema executa automaticamente algumas funcionalidades, como o envio de notificações referentes a eventos importantes, entre eles vencimentos de parcelas e previsões de nascimento de bezerros. Essas ações não dependem da intervenção do usuário, sendo realizadas por mecanismos internos da aplicação.

A Figura 2 apresenta, portanto, as principais funcionalidades do sistema e a forma como cada ator interage com elas, evidenciando uma estrutura voltada à otimização da gestão de pequenas propriedades rurais e ao aumento da eficiência nas atividades cotidianas.

Figura 2 – Diagrama de Casos de Uso



Fonte: Autoria própria (2025).

A seguir, é apresentada a expansão dos principais casos de uso do sistema. Os casos de uso definidos como “Manter”, referem-se aos cadastros com as operações de inclusão, alteração, exclusão e consulta. Essas ações podem ser definidas com a sigla “*CRUD*”: *Create, Read, Update e Delete.* O Quadro 4 ilustra o caso de uso específico para gerenciar contas a receber no sistema, descrevendo suas interações, fluxos de trabalho e os principais requisitos associados.

Quadro 4 - Manter Contas a Receber

Caso de Uso: Manter contas a receber (CRUD) **Descrição**: Este caso de uso refere-se às operações de criação, consulta, edição e exclusão de contas a receber no sistema. **Evento Iniciador**: Necessidade de cadastrar, consultar, editar ou excluir uma conta a receber no sistema.
Atores: Administrador, Funcionário
Pré-condições: 1. Usuário autenticado no sistema. 2. Categoria (plano de contas) cadastrada. 3. Cliente cadastrado.
Pós-condições:

1. Conta a receber registrada com sucesso no sistema.
Sequência típica de eventos (Fluxo Principal): 1. [IN] O usuário acessa o menu "Contas a Receber". 2. [IN] O usuário seleciona a opção desejada: incluir, consultar, editar ou excluir uma conta a receber. 3. [IN] O usuário confirma a operação (botão “salvar”). 4. [OUT] O sistema valida os dados e confirma a operação com uma mensagem de sucesso.
Tratamento de Exceções e Variantes: Exceção 3A: Dados incorretos ou campos obrigatórios não preenchidos 3A.1 [OUT] O sistema informa que algum campo não foi preenchido ou contém um valor inválido. 3A.2 [IN] O usuário revisa os dados e corrige as informações. 3A.3 Retorna ao passo 3.

Fonte: Autoria própria (2025).

O Quadro 5, descreve o caso de uso relacionado às operações de *CRUD* para manter as informações do rebanho de gado.

Quadro 5 - Manter Rebanho de Gado

Caso de Uso: Manter rebanho de gado (CRUD) **Descrição**: Este caso de uso refere-se às operações de criação, consulta, edição e exclusão de registros de animais no sistema, incluindo informações como peso, raça, idade, vacinas e previsões de nascimento. **Evento Iniciador**: Necessidade de cadastrar, consultar, editar ou excluir um animal no sistema.
Atores: Administrador, Funcionário
Pré-condições: 1. Usuário autenticado no sistema. 2. Informações sobre o animal disponíveis para cadastro.
Pós-condições: 1. Registro do animal atualizado com sucesso no sistema.
Sequência típica de eventos (Fluxo Principal): 1. [IN] O usuário acessa o menu "Rebanho". 2. [IN] O usuário seleciona a opção desejada: incluir, consultar, editar ou excluir um animal. 3. [IN] O usuário confirma a operação. 4. [OUT] O sistema valida os dados e confirma a operação com uma mensagem de sucesso.
Tratamento de Exceções e Variantes: Exceção 3A: Dados incorretos ou campos obrigatórios não preenchidos 3A.1 [OUT] O sistema informa que algum campo não foi preenchido ou contém um valor inválido. 3A.2 [IN] O usuário revisa os dados e corrige as informações. 3A.3 Retorna ao passo 3.

Fonte: Autoria própria (2025).

O Quadro 6 descreve o caso de uso relacionado à geração e exportação de relatórios gerenciais e financeiros no sistema, permitindo que o usuário Administrador visualize e extraia informações relevantes para a gestão da propriedade.

Quadro 6 - Gerar e Exportar Relatórios

Caso de Uso: Gerar e exportar relatórios **Descrição**: Este caso de uso refere-se à geração e exportação de relatórios no sistema, permitindo que o usuário administrador consulte dados da propriedade, como dados financeiros, controle de

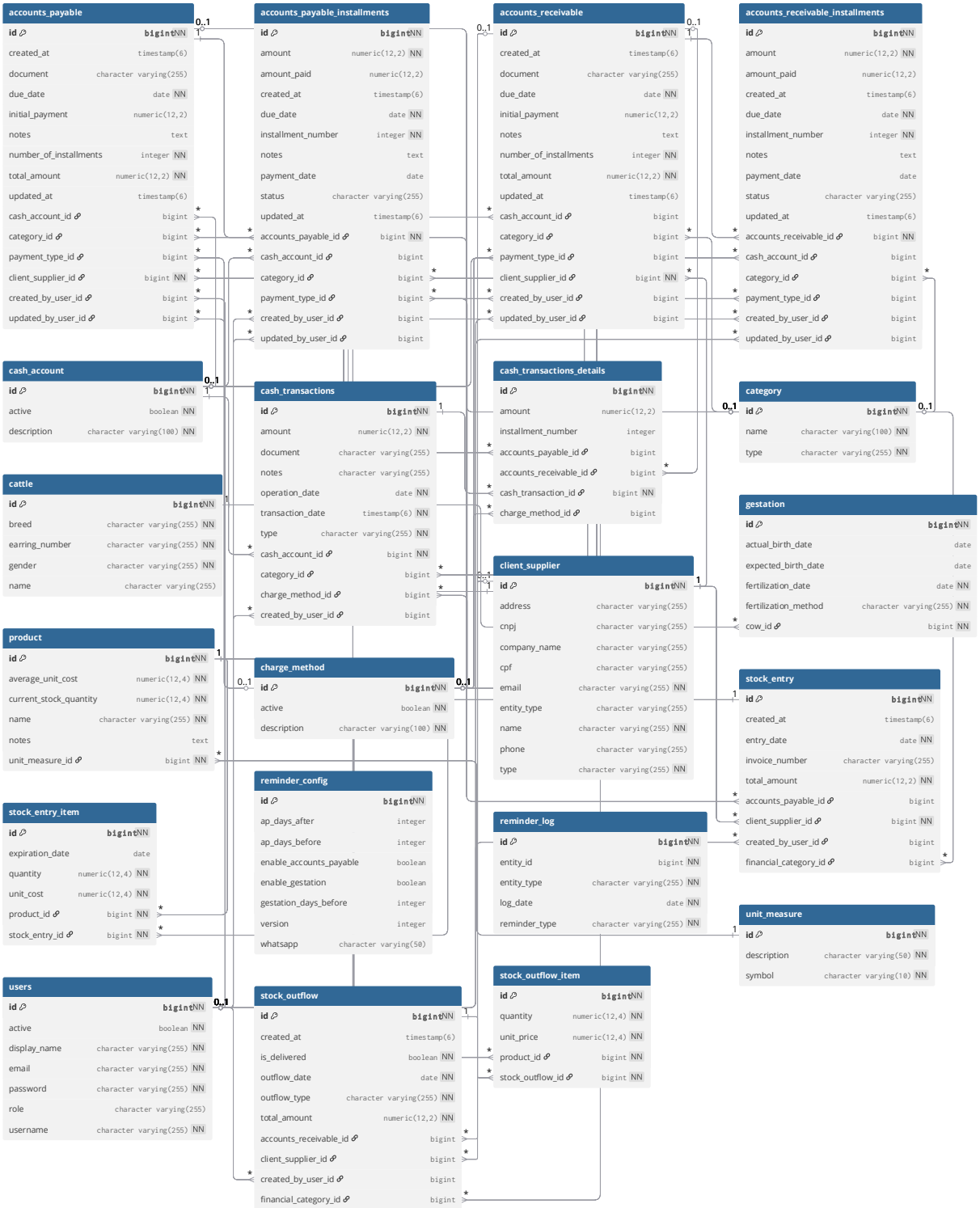
estoque e movimentação do rebanho, com a opção de exportação para formatos como *Portable Document Format* (PDF) e Excel *Open XML Spreadsheet* (XLSX). **Evento Iniciador**: Necessidade de visualizar ou exportar um relatório gerencial.
Atores: Administrador
Pré-condições: 1. Usuário autenticado no sistema. 2. Dados cadastrados no sistema para a geração do relatório.
Pós-condições: 1. Relatório gerado e disponibilizado para visualização ou exportação.
Sequência típica de eventos (Fluxo Principal): 1. [IN] O usuário acessa o menu "Relatórios". 2. [IN] O usuário seleciona o tipo de relatório desejado (financeiro, estoque, rebanho etc.). 3. [IN] O usuário define os filtros e critérios de geração do relatório. 4. [IN] O usuário clica em "Gerar". 5. [OUT] O sistema processa a solicitação e exibe o relatório gerado na tela. 6. [IN] O usuário pode optar por exportar o relatório em PDF ou Excel. 7. [OUT] O sistema gera o arquivo no formato escolhido e disponibiliza o download.
Tratamento de Exceções e Variantes: Exceção 4A: Falta de dados para o relatório 4A.1 [OUT] O sistema informa que não há dados suficientes para gerar o relatório. 4A.2 Retorna ao passo 3.

Fonte: Autoria própria (2025).

4.2.3 Diagrama de Entidade-Relacionamento

A Figura 3 mostra o diagrama de entidade-relacionamento, que fornece uma representação dos relacionamentos entre as tabelas do banco de dados. No diagrama, as tabelas são representadas como entidades, enquanto os relacionamentos entre elas são indicados por meio de linhas e conectores.

Figura 3 - Diagrama de Entidade Relacionamento



Fonte: Autoria própria (2025).

A tabela “users” armazena as informações dos usuários do sistema, incluindo nome, e-mail, credenciais de acesso e status de atividade. Esses dados são utilizados para realizar a autenticação e identificar qual usuário realizou determinadas operações, permitindo controle de acesso e rastreabilidade interna.

A tabela “*client_supplier*” registra os dados dos clientes e fornecedores utilizados nas operações financeiras e de estoque, contendo informações como nome, tipo de entidade (pessoa física ou jurídica), documento (CPF/CNPJ) e dados para contato. Essa tabela é utilizada como referência tanto para transações de compra quanto de venda.

As tabelas “*accounts_payable*” e “*accounts_receivable*” armazenam, respectivamente, os lançamentos de contas a pagar e contas a receber. Cada registro contém informações como data de vencimento, valor total, número de parcelas, observações e o responsável pela criação do lançamento. Ambas possuem relacionamento com clientes/fornecedores, categorias financeiras, forma de pagamento e conta de caixa onde a operação está vinculada.

Para o controle parcelado, o sistema utiliza as tabelas “*accounts_payable_installments*” e “*accounts_receivable_installments*”, onde cada parcela é registrada individualmente com seu número, data de vencimento, valor devido, valor pago, data de pagamento e status (Pendente, Pago ou Cancelado). Dessa forma, é possível acompanhar com precisão o andamento de pagamentos e recebimentos.

A tabela “*cash_account*” representa as contas de caixa ou contas bancárias cadastradas no sistema. Já a tabela “*cash_transactions*” registra todas as movimentações financeiras realizadas nessas contas, indicando se a operação foi de entrada ou saída, seu valor, data e categoria. Para vincular transações a parcelas específicas, utiliza-se a tabela “*cash_transactions_details*”, que detalha o relacionamento entre a movimentação financeira e seu respectivo lançamento ou parcela.

As tabelas “*category*” e “*charge_method*” armazenam, respectivamente, as categorias financeiras utilizadas na classificação das movimentações e os métodos utilizados para pagamento ou recebimento, permitindo a padronização e posterior análise financeira.

No controle do rebanho, a tabela “*cattle*” armazena os dados dos animais, como raça, sexo, identificação por brinco e, opcionalmente, nome. A tabela “*gestation*” registra as gestações das matrizes, contendo informações sobre a data da fertilização, método utilizado (monta natural ou inseminação artificial) e previsão de nascimento, contribuindo para o planejamento da produção.

Para o controle de estoque, a tabela “*product*” armazena os produtos cadastrados, juntamente com sua unidade de medida e custo médio. A entrada de produtos é registrada na tabela “*stock_entry*”, e seus itens são detalhados na “*stock_entry_item*”, contendo quantidade, custo unitário e informações adicionais. De forma semelhante, a saída de itens é controlada pelas tabelas “*stock_outflow*” e “*stock_outflow_item*”, que registram consumos e vendas com seus respectivos totais e itens movimentados.

Por fim, o sistema também conta com as tabelas “*reminder_config*” e “*reminder_log*”, responsáveis pelo controle e registro de lembretes automáticos, como vencimento de contas e previsão de nascimento, auxiliando na organização das atividades da propriedade.

4.3 Apresentação do sistema

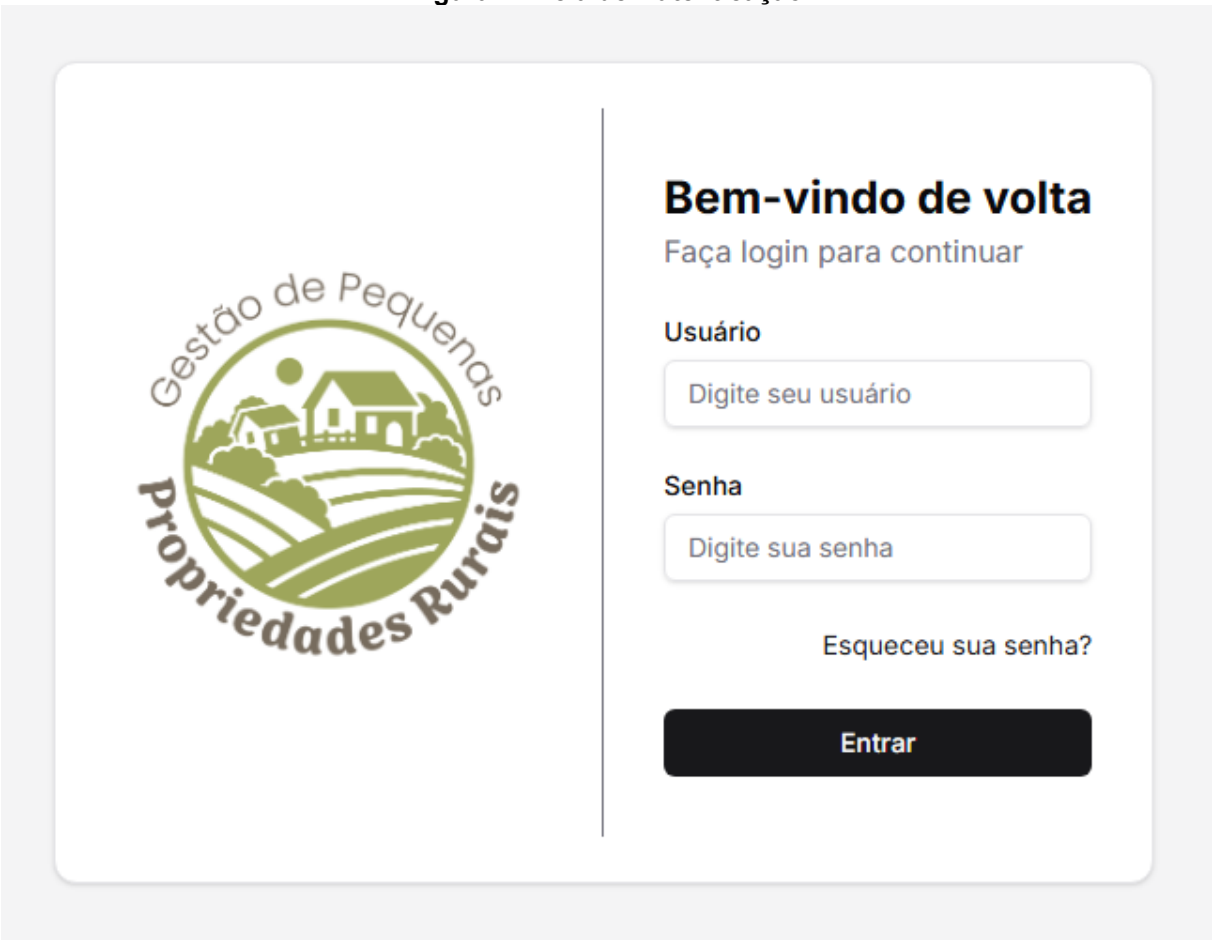
A seguir, serão apresentadas as principais telas e funcionalidades desenvolvidas no sistema desenvolvido.

Para a construção das interfaces, adotou-se o *shadcn/ui*, uma biblioteca de componentes baseada em React que fornece elementos visuais reutilizáveis e altamente customizáveis. Diferente de bibliotecas tradicionais que oferecem componentes prontos, o *shadcn/ui* distribui o código-fonte dos componentes para o projeto, permitindo que o desenvolvedor tenha controle total sobre sua estrutura, estilização e comportamento.

Essa abordagem favorece uma padronização visual em todas as telas do sistema, ao mesmo tempo em que permite ajustes finos conforme as necessidades específicas da aplicação. A estilização dos componentes da biblioteca é realizada predominantemente utilizando Tailwind CSS, o que torna o processo mais ágil e consistente, além de facilitar a manutenção futura. Dessa forma, o sistema apresenta uma identidade visual coerente, moderna e alinhada às boas práticas de interface e experiência do usuário.

A Figura 4 apresenta a tela de autenticação do sistema, na qual o usuário deve informar seu nome de usuário e sua senha cadastrados. Após preencher os campos, o acesso ao sistema é realizado por meio do botão “Entrar”.

Figura 4 - Tela de Autenticação

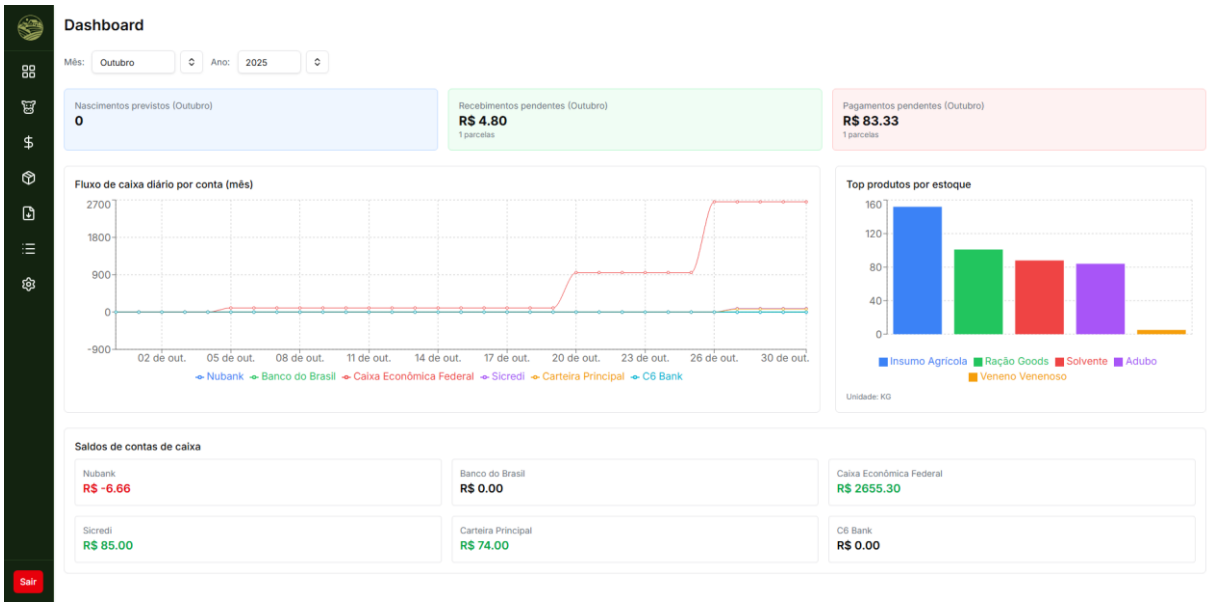


Fonte: Autoria própria (2025).

Após a autenticação ser realizada com sucesso, o usuário é redirecionado para a tela inicial do sistema, onde é exibido um *dashboard* contendo as principais informações para apoio à tomada de decisão. Nesta tela, são apresentados o saldo total das contas cadastradas, o número de nascimentos previstos para o mês selecionado, além de um resumo dos recebimentos e pagamentos pendentes no período. Também é exibido o estoque atual dos produtos registrados no sistema, conforme ilustrado na Figura 5.

Para a construção dos gráficos presentes no *dashboard*, foi utilizada a biblioteca Recharts, uma ferramenta baseada em componentes React que permite criar visualizações de dados de maneira simples e flexível. Sua utilização possibilita a exibição clara e dinâmica das informações, garantindo melhor compreensão dos indicadores por parte do usuário e contribuindo para uma experiência visual mais intuitiva e agradável.

Figura 5 - Tela de *Dashboard*



Fonte: Autoria própria (2025).

No canto superior esquerdo da tela, encontram-se os filtros de Mês e Ano, os quais, por padrão, exibem os dados referentes ao mês e ano corrente. Ao modificar esses filtros, as informações apresentadas no *dashboard* são automaticamente atualizadas, conforme ilustrado na Figura 6.

Figura 6 - Demonstração do Filtro do *Dashboard*

Fonte: Autoria própria (2025).

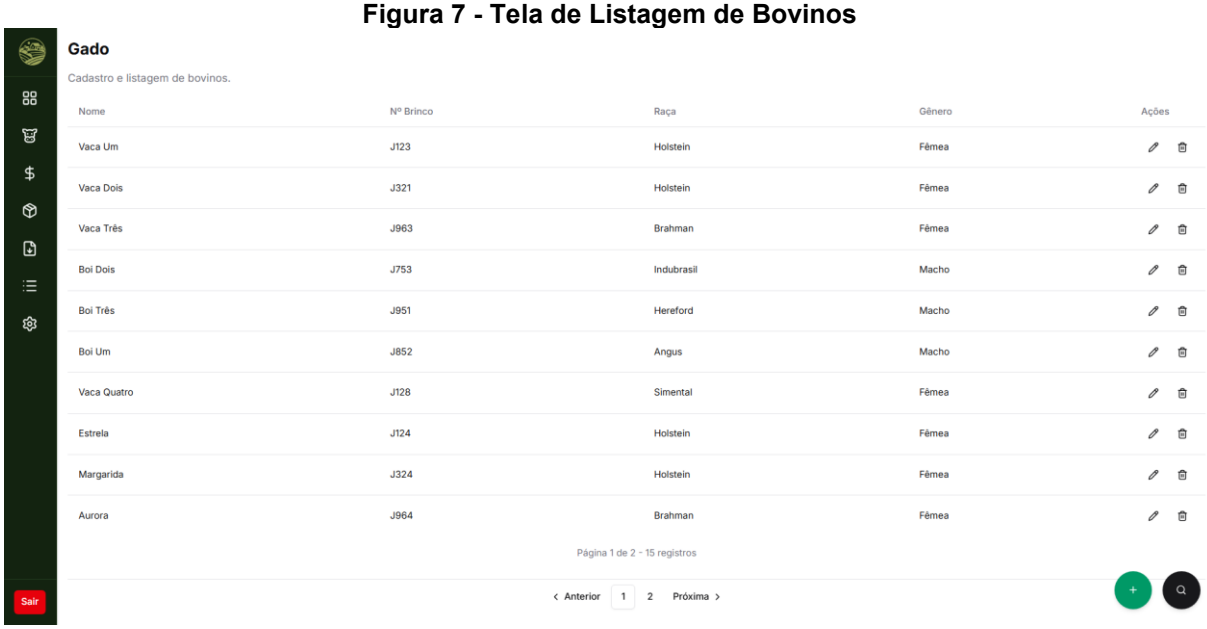
As Figuras 5 e 6 também apresentam a visualização de todos os gráficos e totalizadores, sendo eles:

- Totalizadores: Na parte de cima da tela, são exibidos três totalizadores (Nascimentos previstos, Recebimentos pendentes e Pagamentos pendentes), que representam, respectivamente, soma dos números de partos previstos, soma total de contas a receber pendentes e a pagar dentro do mês e ano filtrados. Na parte inferior da tela, também é mostrado outros totalizados com o saldo histórico de cada conta de caixa.
- Fluxo de caixa diário por conta (Gráfico de Linhas): Exibe a evolução diária do saldo de cada conta de caixa cadastrada, sendo uma linha por caixa, dentro do mês e ano filtrados.
- Top produtos por estoque (Gráfico de Barras Verticais): Mostra os cinco itens com mais saldo em estoque atualmente.

A análise leva em conta os dados filtrados junto aos registros históricos, garantindo que os totalizadores e gráficos apresentem uma visão precisa e consolidada da movimentação financeira e do estoque.

A figura 7 apresenta a tela de cadastro de gado, na qual é possível visualizar todos os bovinos registrados no sistema por meio de um *grid*, que, assim como todas as telas de listagem, utiliza um componente shadcn/ui. Em cada linha da tabela são disponibilizados botões específicos para edição e exclusão do respectivo registro, bem como 2 botões flutuantes para inclusão (ícone de “+”) e para filtro (ícone de lupa), que também estão presentes na maioria das telas.

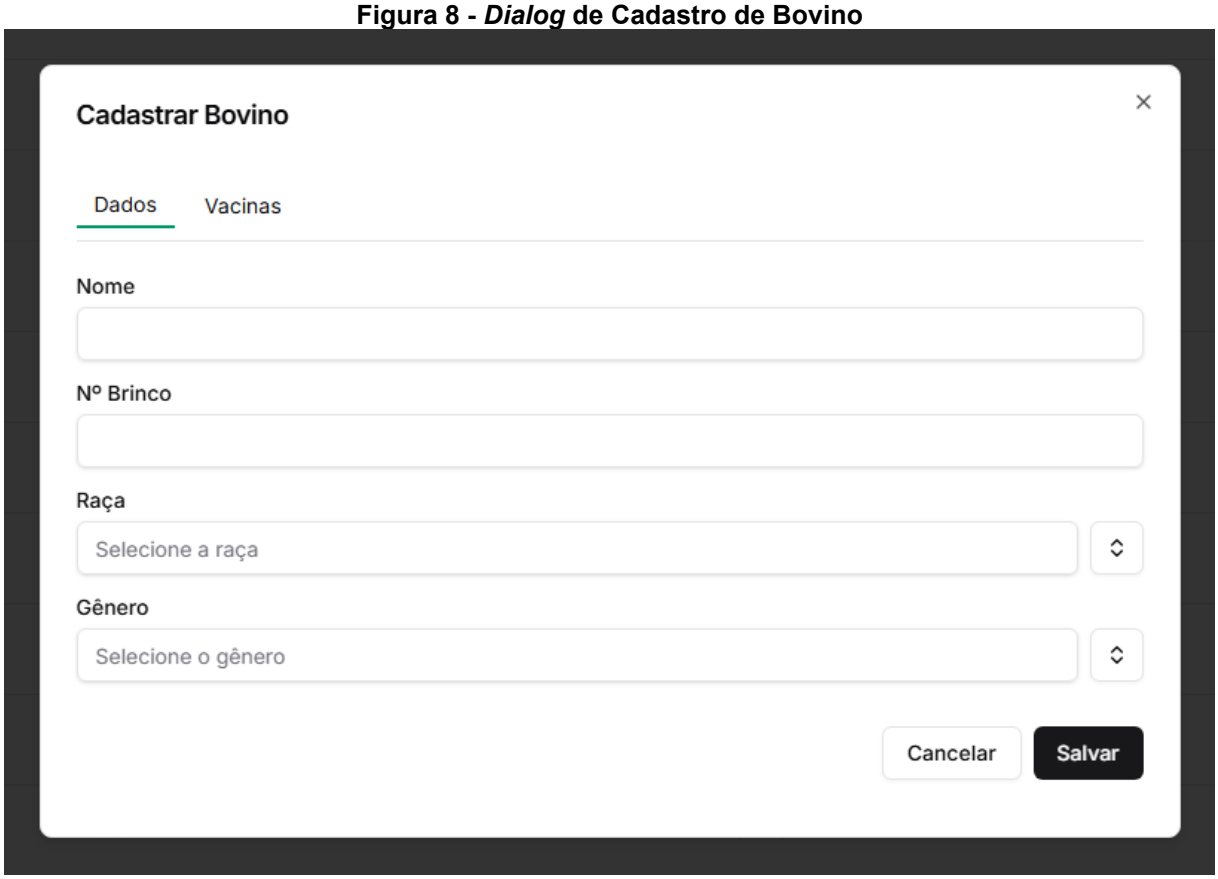
Figura 7 - Tela de Listagem de Bovinos



Fonte: Autoria própria (2025).

Ao clicar no botão de inclusão, o sistema abre um componente do tipo *Dialog*, solicitando o preenchimento de um formulário contendo campos como Nome, Número do Brinco, Raça e Gênero, conforme mostra a Figura 8. O shadcn/ui não disponibiliza um componente específico classificado como *Modal*; em vez disso, a biblioteca utiliza o conceito de *Dialog* para representar janelas sobrepostas à interface. Assim, todos os diálogos modais presentes nas telas do sistema foram desenvolvidos com base nesse componente, garantindo consistência na experiência de interação e na identidade visual. O número do brinco, no contexto de propriedades rurais, é utilizado como identificador individual do animal, permitindo seu reconhecimento de forma rápida e precisa no rebanho, além de facilitar o controle de informações sanitárias, produtivas e de manejo.

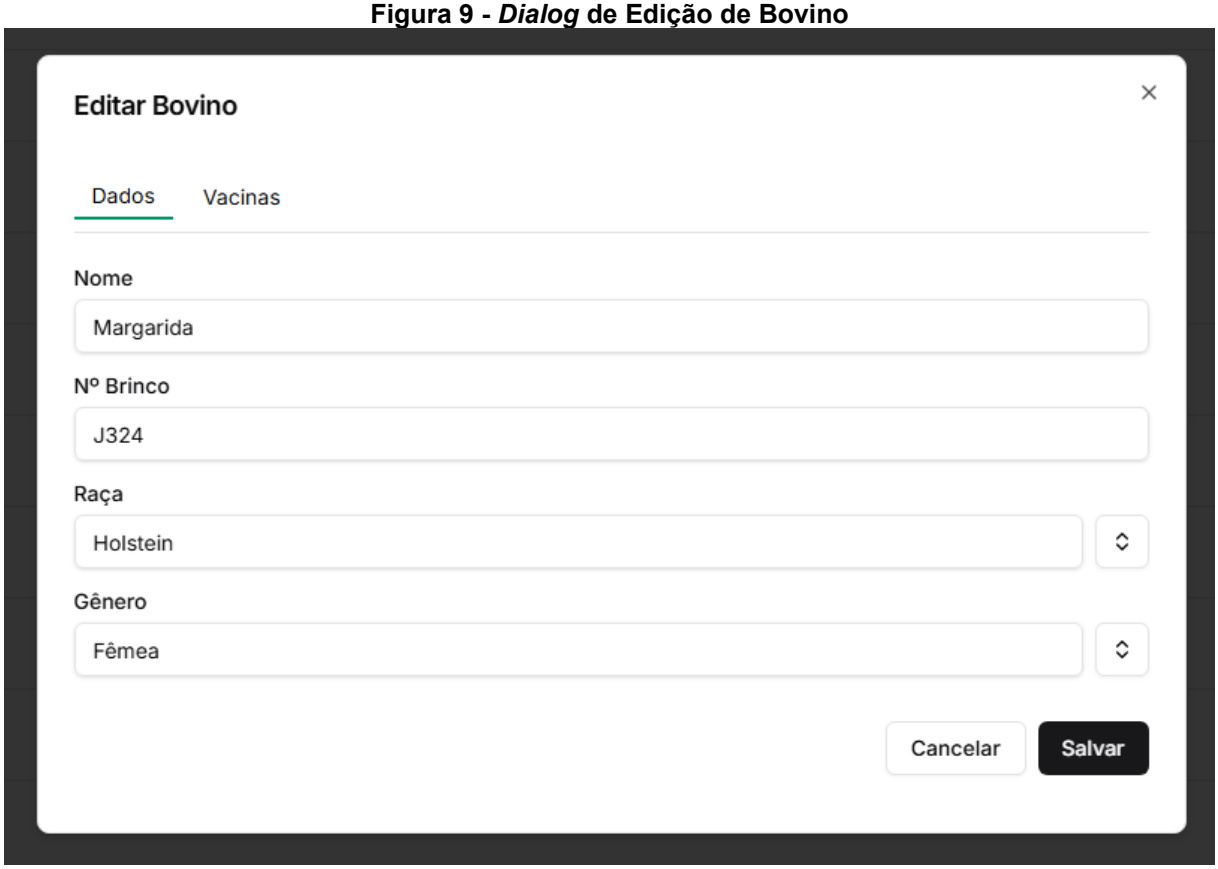
Figura 8 - *Dialog* de Cadastro de Bovino



Fonte: Autoria própria (2025).

Ao selecionar o botão de edição, representado pelo ícone de lápis, o sistema exibe um *dialog* contendo os campos previamente preenchidos com as informações atuais do animal, permitindo sua alteração na aba “Dados”. Esse processo pode ser observado na Figura 9.

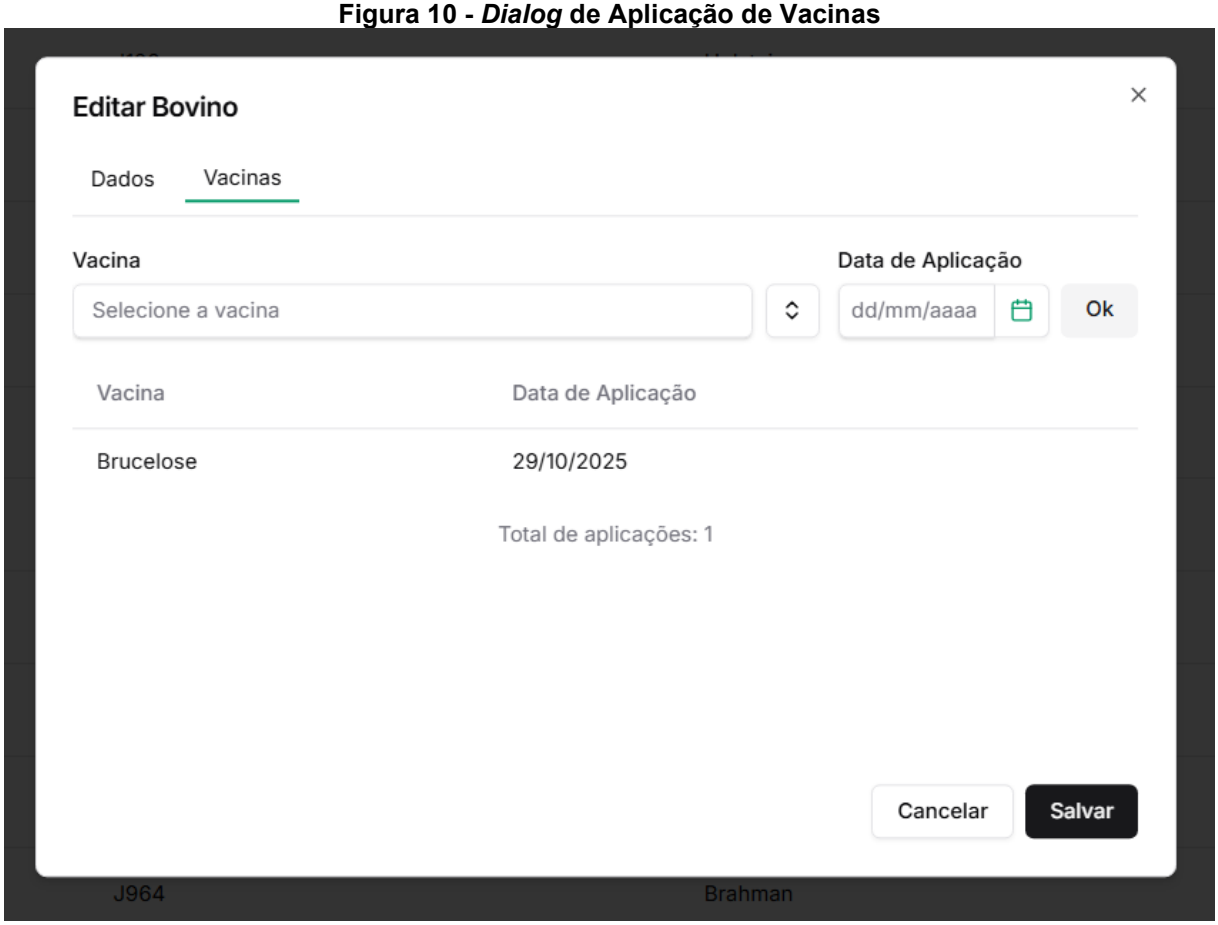
Figura 9 - *Dialog* de Edição de Bovino



Fonte: Autoria própria (2025).

Nessa mesma tela de edição, existe a aba “Vacinas”, onde é realizado todo o acompanhamento da vacinação naquele bovino. Na parte de cima, o usuário escolhe dentre os medicamentos cadastrados previamente e a data da aplicação, já na parte de baixo, o usuário acompanha todas as aplicações, conforme Figura 10.

Figura 10 - *Dialog* de Aplicação de Vacinas



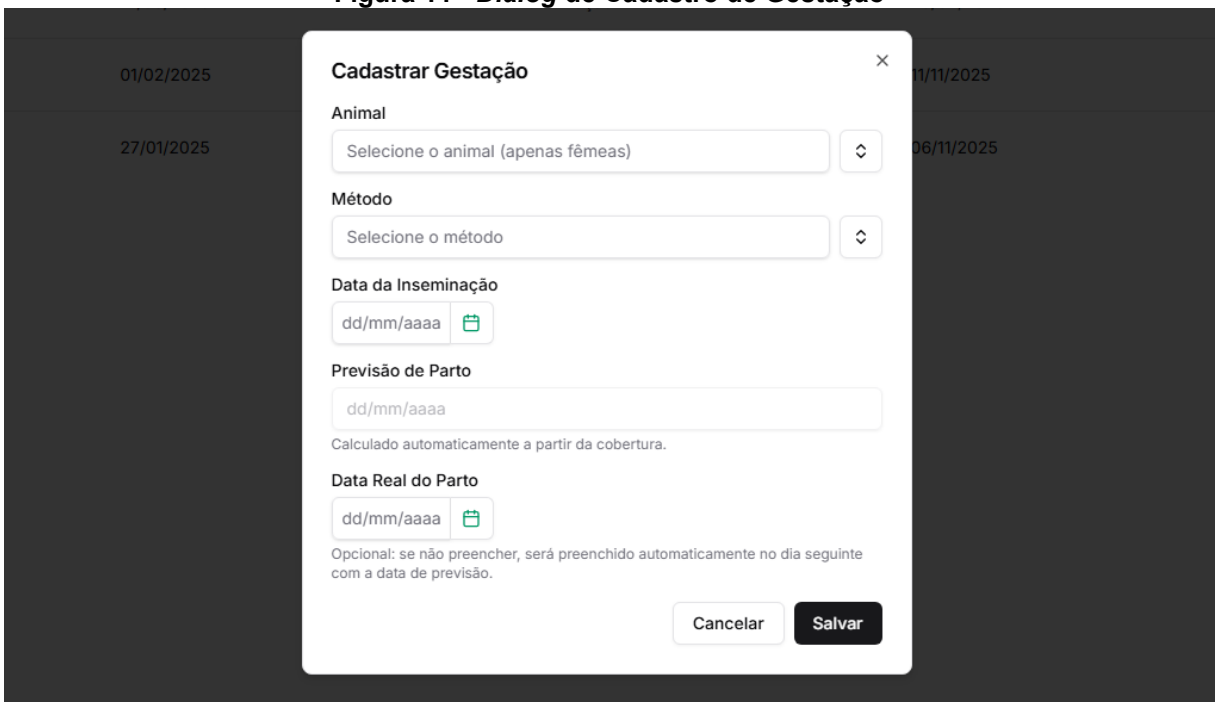
Fonte: Autoria própria (2025).

A tela do módulo de Gestação foi desenvolvida com a finalidade de facilitar o acompanhamento das gestações em andamento, atendendo ao objetivo de visualizar e prever a taxa de natalidade do rebanho bovino, possibilitando ainda a emissão de relatórios detalhados contendo informações como a data de concepção ou de inseminação (em casos de reprodução assistida). Nessa tela, é apresentado um *grid* contendo todas as gestações previamente registradas, permitindo identificar rapidamente quais fêmeas estão com gestação ativa, além de visualizar outras informações relevantes diretamente na listagem. Assim como nas demais telas de visualização de cadastros, cada registro conta com ícones de edição e exclusão, mantendo a padronização da interface do sistema.

A Figura 11 apresenta os campos disponíveis no formulário de cadastro de uma gestação. O campo “Animal” é filtrado para exibir apenas fêmeas, visto que somente elas podem passar pelo processo gestacional. O campo “Método” permite selecionar entre “Monta Natural”, que corresponde ao acasalamento direto entre o touro e a vaca, e “Inseminação Artificial”.

O formulário também inclui os campos “Data da Inseminação” e “Data Real do Parto”. O campo “Previsão de Parto” possui preenchimento automático, sendo calculado pelo sistema ao somar 283 dias à data da inseminação, não sendo possível sua edição manual. Caso o usuário não informe a “Data Real do Parto”, o sistema realizará seu preenchimento de forma automática um dia após a data prevista, utilizando como referência a previsão calculada.

Figura 11 - *Dialog* de Cadastro de Gestação



Fonte: Autoria própria (2025).

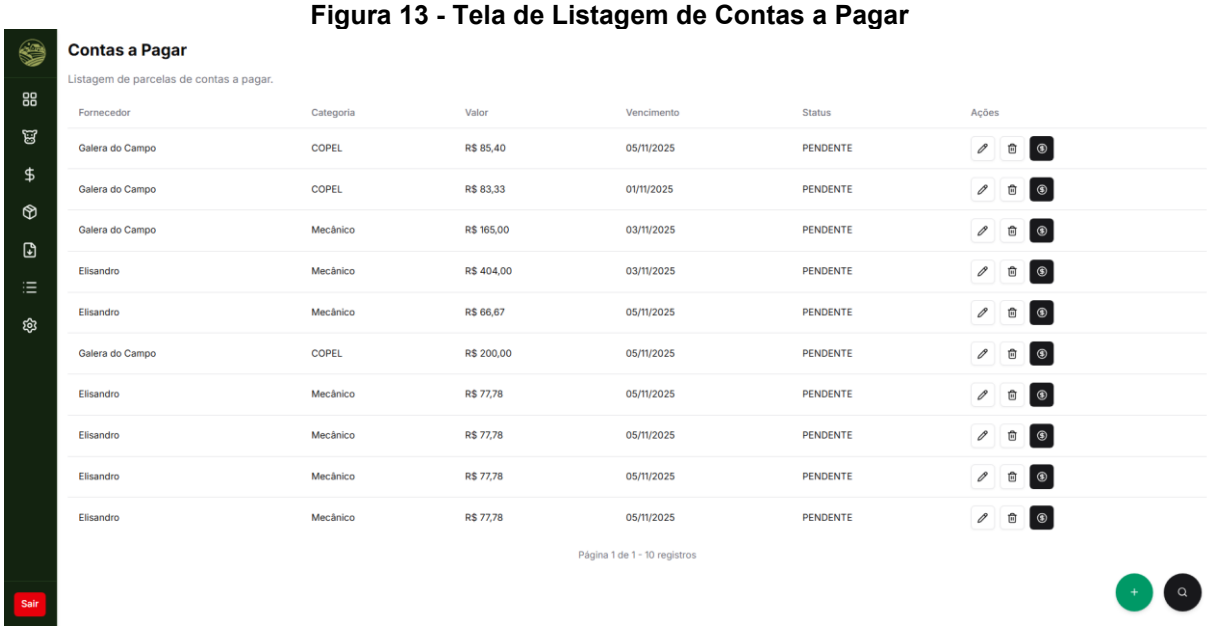
Após o lançamento de gestações, é possível realizar a geração do relatório de gestações. Esse relatório permite filtrar por animal (fêmea) ou gerar de todas as vacas de uma única vez, como ilustrado na Figura 12. Também conta com filtro de “Método de fertilização” e de datas, como inseminação e previsão de nascimento. Todos os relatórios do sistema são gerados nos formatos PDF ou Excel.

Figura 12 - Tela de Emissão de Relatórios de Gestação

Fonte: Autoria própria (2025).

Ao acessar a tela de Contas a Pagar, o sistema exibe, por padrão, no *grid*, todas as parcelas pendentes referentes ao mês vigente. Essa funcionalidade tem como objetivo auxiliar o usuário na visualização rápida dos compromissos financeiros atuais, evitando esquecimentos. Para isso, são aplicados automaticamente filtros ao carregar a tela, os quais podem ser alterados posteriormente na seção de filtros, caso o usuário deseje realizar consultas mais específicas. Na listagem, além dos botões de edição e exclusão, também está disponível o botão para realizar a quitação das parcelas, conforme ilustrado na Figura 13.

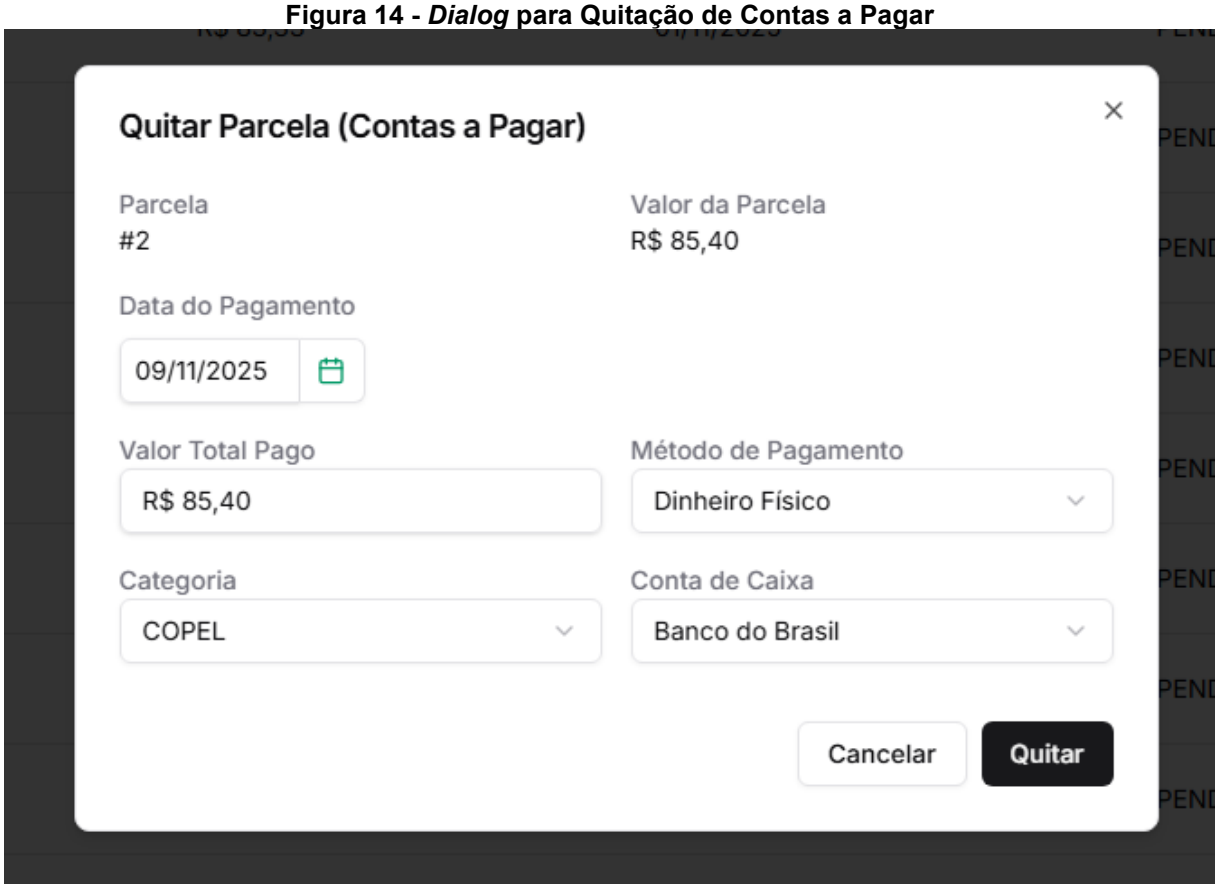
Figura 13 - Tela de Listagem de Contas a Pagar



Fonte: Autoria própria (2025).

Ao clicar no botão de quitação, o sistema abre um *dialog* contendo os campos necessários para concluir o pagamento, sendo eles: “Data do Pagamento”, “Valor Total Pago”, “Método de Pagamento”, “Categoria” e “Caixa”, conforme apresentado na Figura 14. Considerando que se trata de uma Conta a Pagar, foi implementada uma regra específica no campo “Categoria”, de modo que sejam exibidas somente categorias do tipo “saída”, que representam retiradas de valores do caixa selecionado. Ao confirmar a operação, o sistema registra automaticamente a movimentação financeira correspondente, alterando o status da parcela para “Pago” e lançando a saída no caixa.

Figura 14 - *Dialog* para Quitação de Contas a Pagar

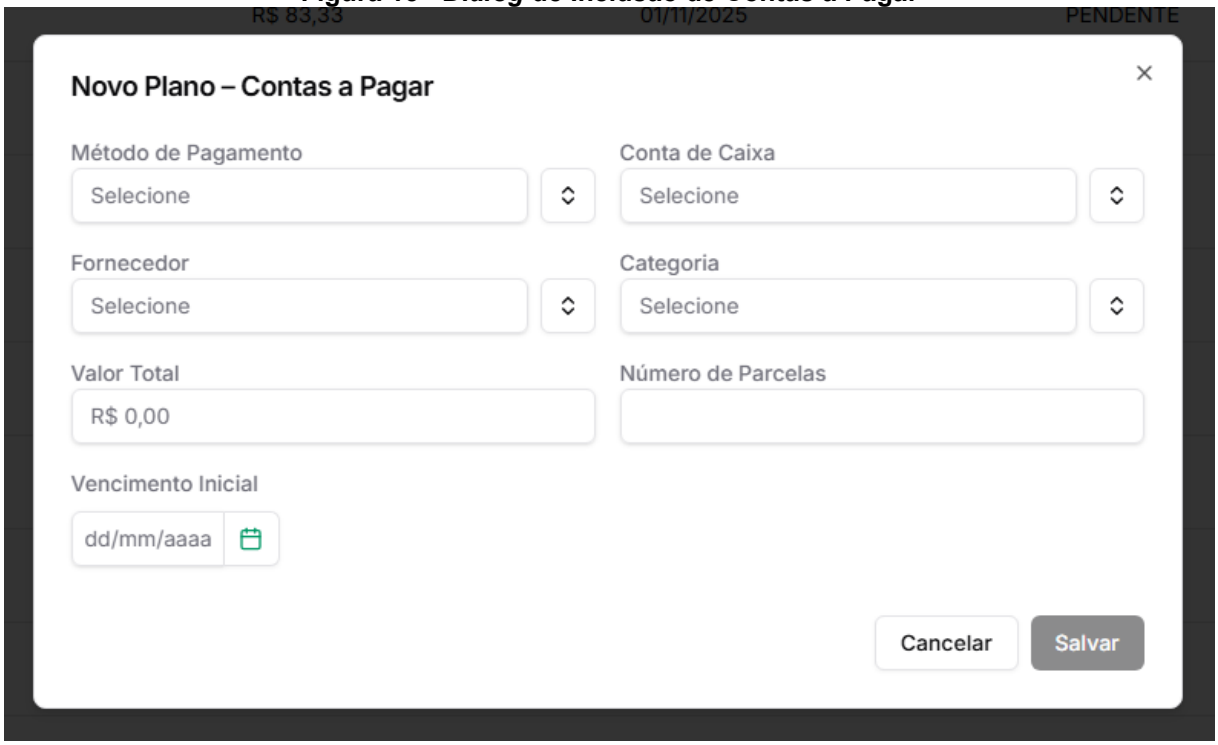


Fonte: Autoria própria (2025).

Na Figura 15 é apresentada a tela de inclusão de um novo plano de contas a pagar. Nela, o usuário encontra campos do tipo *combobox* para seleção de “Método de Pagamento”, “Conta de Caixa”, “Fornecedor” e “Categoria”, além dos campos de preenchimento de valores. Entre eles, destaca-se o campo “Valor Total”, onde é informado o valor integral do plano, e o campo “Número de Parcelas”, utilizado para definir o parcelamento desejado. Também está presente o campo de data

“Vencimento Inicial”, responsável por determinar a data de vencimento da primeira parcela, que servirá de referência para o cálculo das datas das demais parcelas nos meses seguintes.

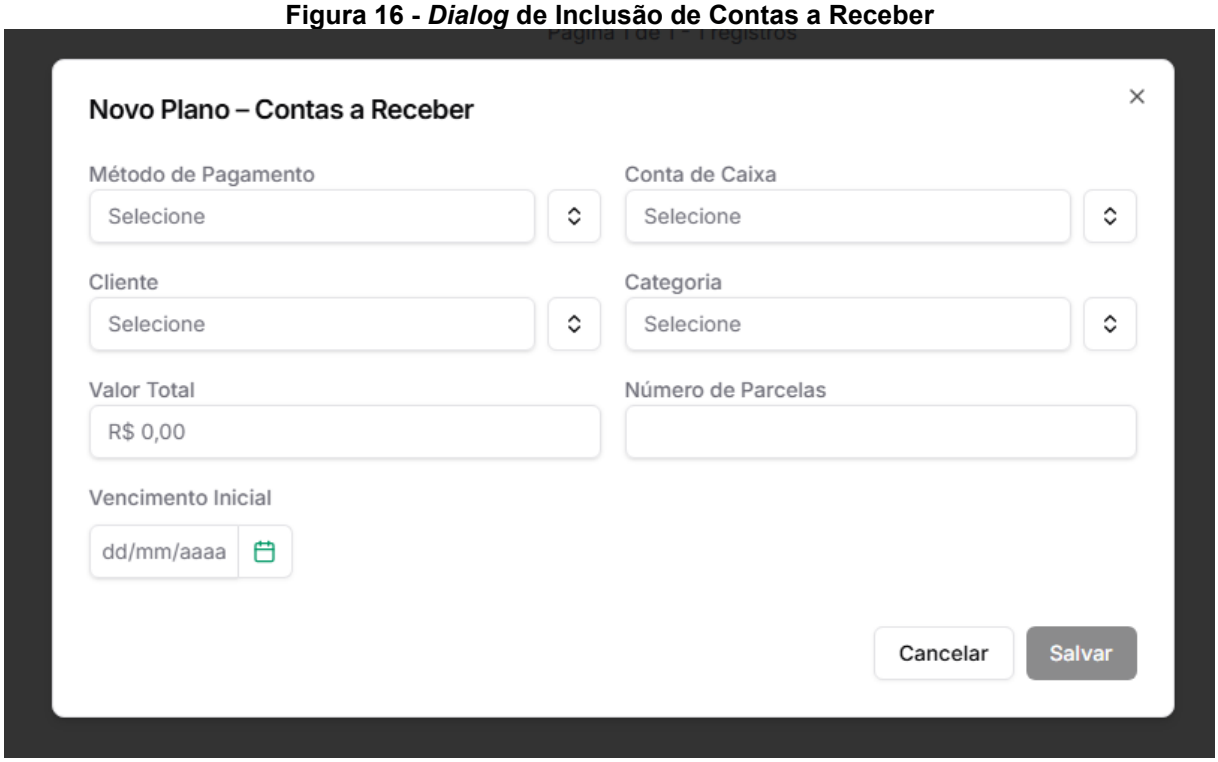
Figura 15 - *Dialog* de Inclusão de Contas a Pagar



Fonte: Autoria própria (2025).

O módulo de Contas a Receber apresenta estrutura, filtros e interface semelhantes ao módulo de Contas a Pagar, incluindo *grid* de listagem, botões de edição, exclusão e registro de quitações, bem como tela para inclusão de novos planos. Conforme Figura 16, as diferenças estão nos campos de “Categoria”, que neste módulo exibe exclusivamente categorias do tipo “entrada”, de modo a representar o recebimento de valores no caixa e no campo de “Clientes”, que não irá mostrar os registros do tipo “Fornecedor”.

Figura 16 - *Dialog* de Inclusão de Contas a Receber

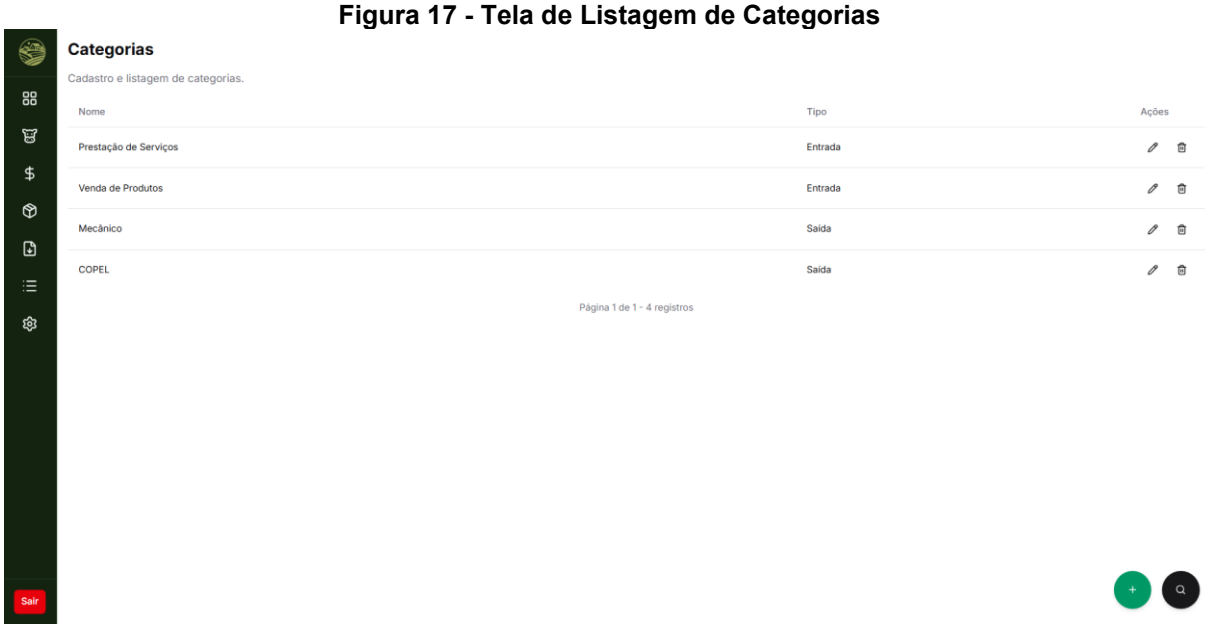


Fonte: Autoria própria (2025).

Os cadastros de contas a pagar e contas a receber estão diretamente vinculados às “Categorias”, que representam as classificações responsáveis por organizar todas as movimentações financeiras da propriedade. Essas categorias agrupam os lançamentos em conjuntos lógicos, permitindo uma análise mais clara dos fluxos de entrada e saída de recursos, além de facilitar a interpretação dos relatórios gerados pelo sistema.

A Figura 17 mostra a tela de gerenciamento de categorias, que segue o mesmo padrão visual e funcional das demais telas do sistema, apresentando um *grid* com a listagem das categorias cadastradas, botões para edição e exclusão de registros do *grid* e os botões flutuantes de inclusão e filtro, mantendo a padronização da interface.

Figura 17 - Tela de Listagem de Categorias



Fonte: Autoria própria (2025).

Ao acionar o botão de inclusão, é exibido um *dialog* contendo apenas dois campos: “Descrição” e “Tipo”, visualizado na Figura 18. O campo “Nome” é utilizado para adicionar uma descrição breve à categoria, enquanto o campo “Tipo” permite selecionar se a categoria será do tipo entrada (utilizada em Contas a Receber) ou saída (utilizada em Contas a Pagar), atendendo ao objetivo que visa possibilitar o cadastro de categorias para controle de entradas e saídas de caixa, permitindo a gestão financeira da propriedade.

Figura 18 - *Dialog* para Inclusão de Categoria

Fonte: Autoria própria (2025).

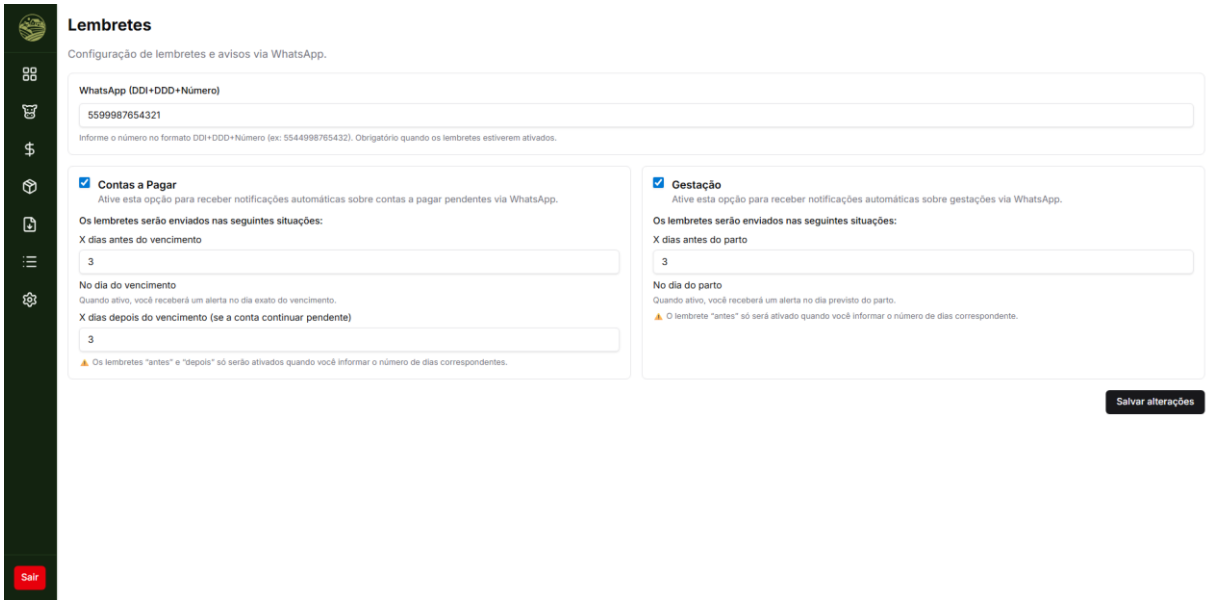
Para atender ao objetivo de envio de alertas sobre vencimentos de contas a pagar e proporcionar um controle mais eficiente, foi desenvolvido um sistema de envio de lembretes sobre contas a pagar pendentes e sobre gestações próximas via WhatsApp. A Figura 19 ilustra a tela completa de notificações, que conta com os seguintes campos:

- WhatsApp (*Input* de número): Campo obrigatório para permitir ativar as opções de recebimento. Deve ser preenchido utilizando o padrão de formatação de número do *WhatsApp*.
- Contas a Pagar (C*heckbox*): Ao marcar a *checkbox* referente à Contas a Pagar, o sistema irá habilitar o envio de notificações em três situações, não sendo necessário nenhuma habilitação adicional:
    - X dias antes do vencimento (*Input* de número): Quando a *checkbox* acima está marcada, se torna obrigatório o preenchimento desse campo. O usuário irá definir quantos dias antes do vencimento receberá as informações referentes à conta.
    - No dia do vencimento (campo apenas informativo): Quando a *checkbox* acima está marcada, o sistema enviará notificações referente às parcelas que vencem no dia da verificação.
    - X dias depois do vencimento (*Input* de número): Quando a *checkbox* acima está marcada, se torna obrigatório o preenchimento desse campo. O usuário irá definir quantos dias após o vencimento receberá as informações referentes à conta.

  Para os três casos, o sistema realiza a verificação se as parcelas em questão continuam pendentes no sistema, ignorando as parcelas que já foram quitadas.
- Gestação (*Checkbox*): Deve ser habilitado quando o usuário deseja receber informações no *WhatsApp* sobre nascimentos de bezerros. Seguindo uma lógica parecida com os envios de Contas a Pagar, o sistema realiza a coleta das datas das gestações lançadas no sistema e envia notificações em duas situações:
    - X dias antes do parto (*Input de número*): Quando a *checkbox* acima está marcada, se torna obrigatório o preenchimento desse campo. O usuário irá definir quantos dias antes da previsão de nascimento receberá as informações referentes à fêmea.

- No dia do parto (campo apenas informativo): Quando a *checkbox* acima está marcada, o sistema enviará notificações referente às gestações que estão previstas para o dia da verificação.

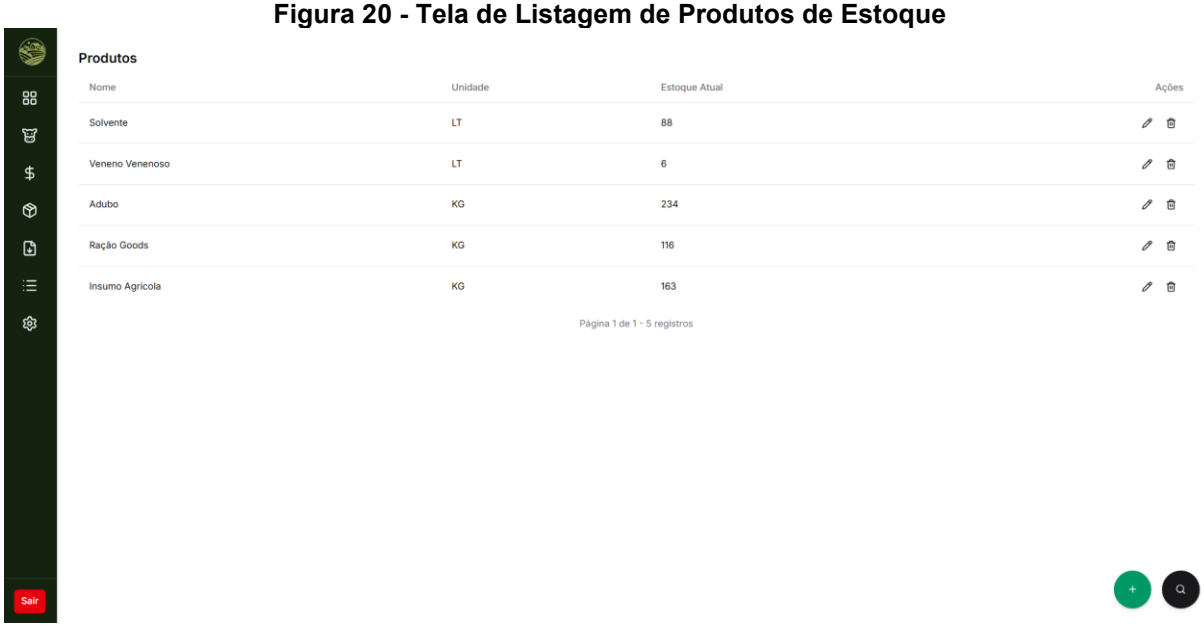
Figura 19 - Tela de Configuração de Lembretes via WhatsApp



Fonte: Autoria própria (2025).

Para atingir o objetivo “possibilitar o controle de estoque de rações e insumos, incluindo registro de uso e venda” e “possibilitar classificação de itens do estoque e emissão de relatórios detalhados de entradas, saídas e saldo”, foi desenvolvido um módulo específico para controle de itens, saldo e movimentações de estoque. Na Figura 20, observa-se a tela de listagem de produtos, onde é apresentado um *grid* com o mesmo padrão das outras telas, mostrando o estoque atual de cada um dos itens, que é calculado automaticamente com base nas entradas e saídas de estoque.

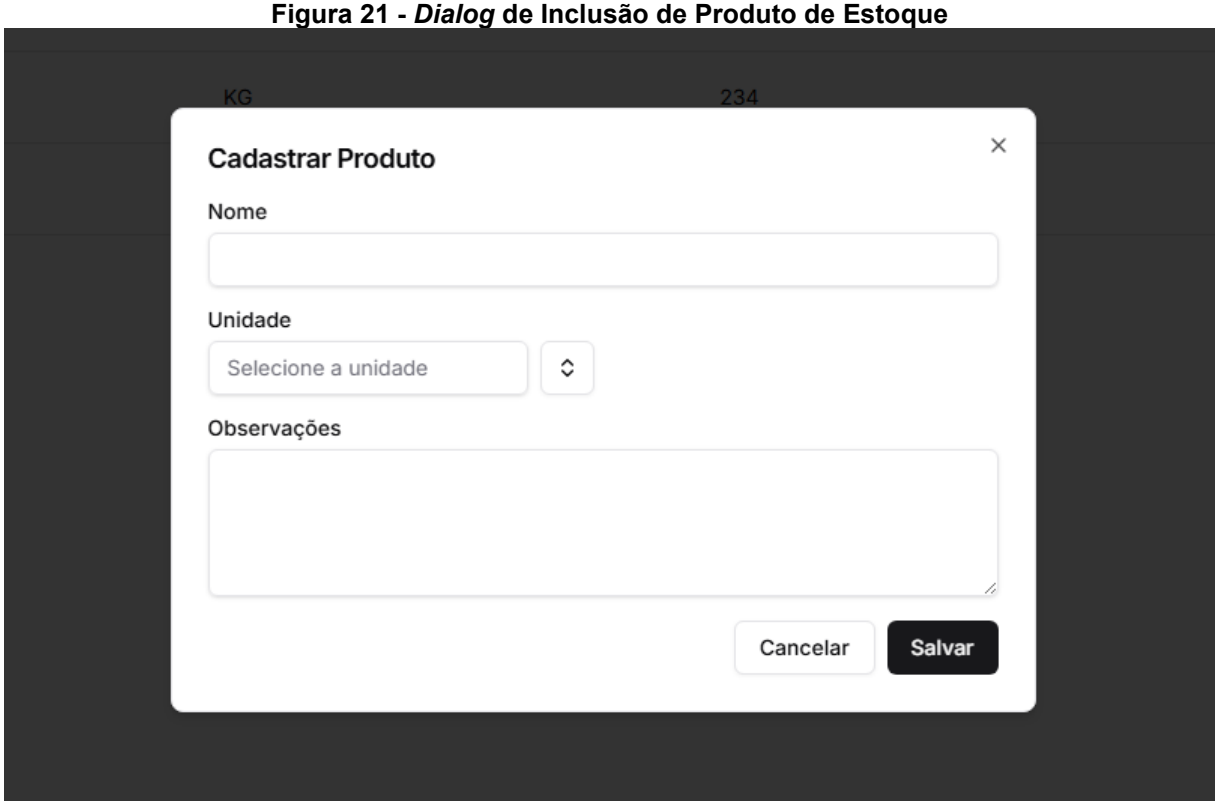
Figura 20 - Tela de Listagem de Produtos de Estoque



Fonte: Autoria própria (2025).

A Figura 21 ilustra o *dialog* de inclusão de produtos, onde é solicitado ao usuário o nome e unidade de medida, além de um campo opcional para observações sobre o item cadastrado.

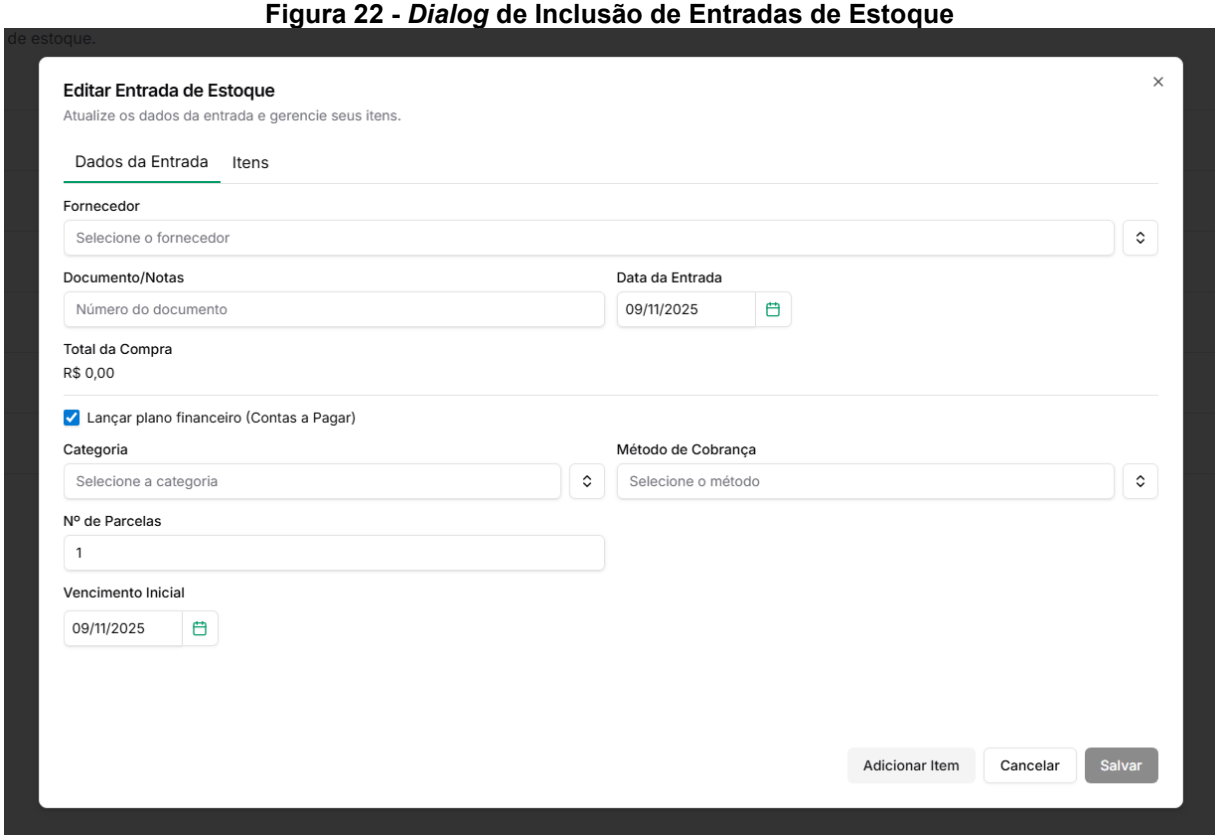
Figura 21 - *Dialog* de Inclusão de Produto de Estoque



Fonte: Autoria própria (2025).

A tela de inclusão de entrada de estoque, representada pela Figura 22, solicita ao usuário os dados do fornecedor, para registrar de onde foi comprado o produto (ou listagem de produtos), bem como um campo opcional para informar o número do documento ou nota fiscal, auxiliando na documentação e rastreabilidade da entrada. Para facilitar o controle financeiro, quando a opção “Lançar plano financeiro (Contas a Pagar)” está marcada, o sistema solicita os dados financeiros para realizar o lançamento de contas a pagar, com o valor calculado automaticamente baseado na soma dos itens da aba “Itens”.

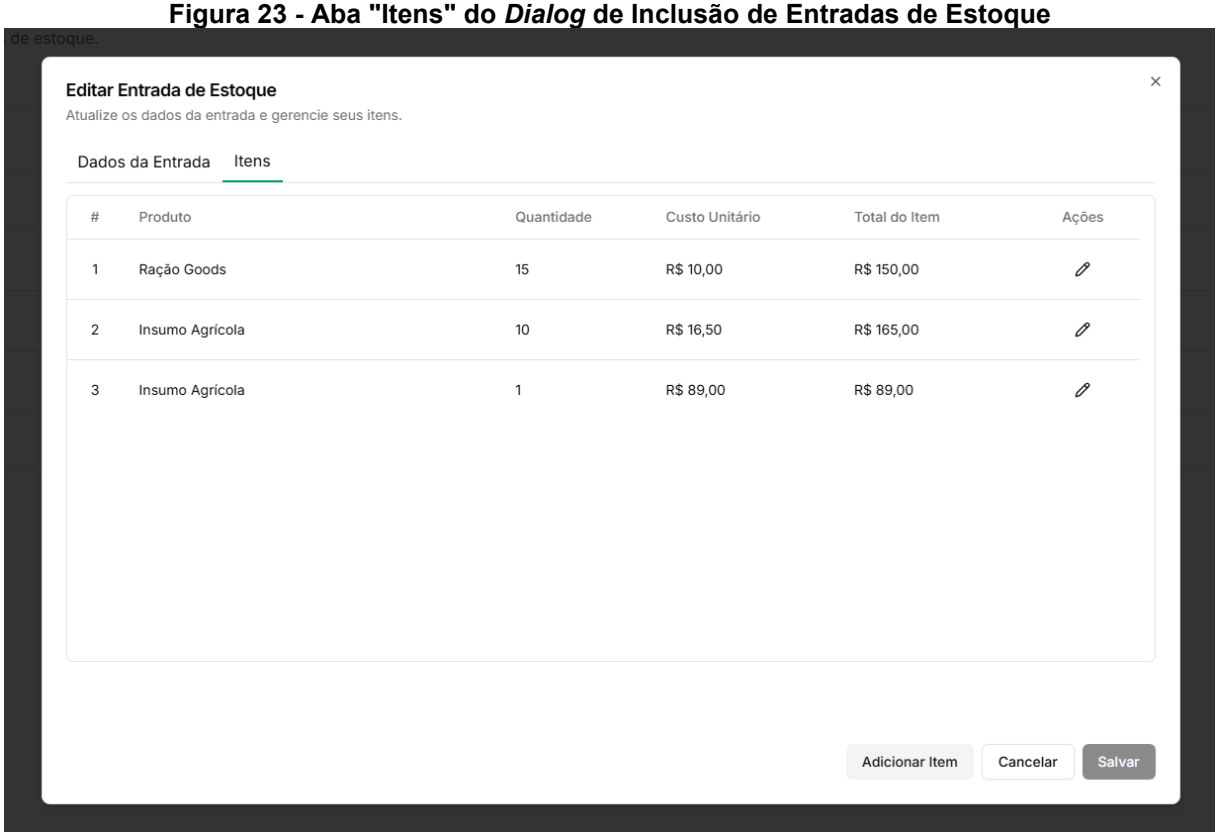
Figura 22 - *Dialog* de Inclusão de Entradas de Estoque



Fonte: Autoria própria (2025).

A aba “Itens”, ilustrada na Figura 23, permite o controle dos produtos adquiridos do fornecedor selecionado. Ao acionar o botão “Adicionar Item”, o sistema insere uma nova linha no *grid*, possibilitando a seleção do produto, a definição da quantidade e o preenchimento do custo unitário. A partir dessas informações, o valor total de cada item é calculado automaticamente, assim como o valor total da compra, obtido pela soma dos valores individuais registrados.

Figura 23 - Aba "Itens" do *Dialog* de Inclusão de Entradas de Estoque



Fonte: Autoria própria (2025).

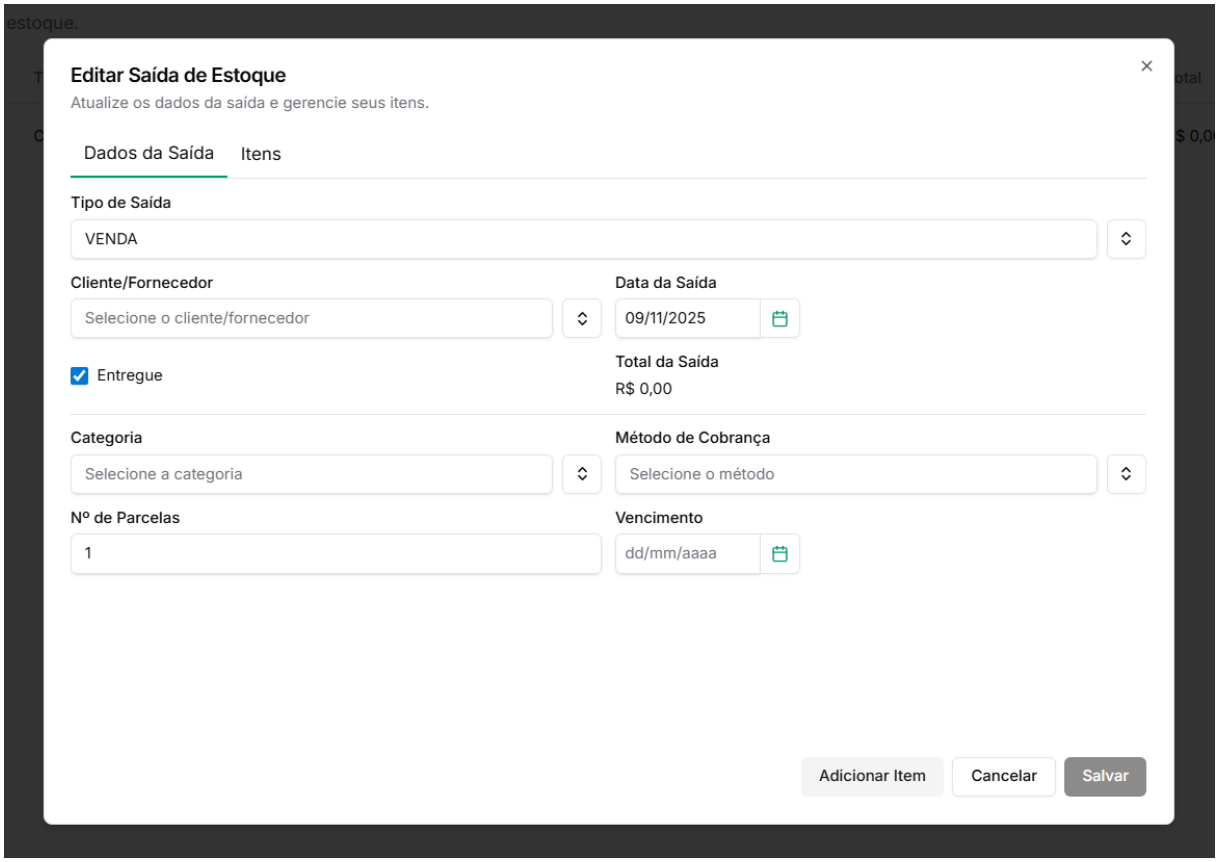
O menu de saídas de estoque apresenta funcionalidades semelhantes ao menu de Entradas de Estoque, diferenciando-se pela necessidade de informar o “Tipo de Saída”, que pode ser “Consumo” ou “Venda”, além da funcionalidade de entrega dos produtos.

Quando o item é marcado como “Entregue”, o sistema realiza automaticamente a baixa do estoque, atualizando o saldo disponível. No caso de

saídas do tipo “Consumo”, o campo “Cliente/Fornecedor” permanece oculto, uma vez que o produto será utilizado internamente na propriedade.

Já nas saídas do tipo “Venda”, ao marcar o item como “Entregue”, o usuário tem a opção de gerar um lançamento financeiro, criando um registro da venda no menu “Contas a Receber”. Essa dinâmica pode ser observada na Figura 24.

Figura 24 - *Dialog* de Saída de Estoque tipo "Venda"

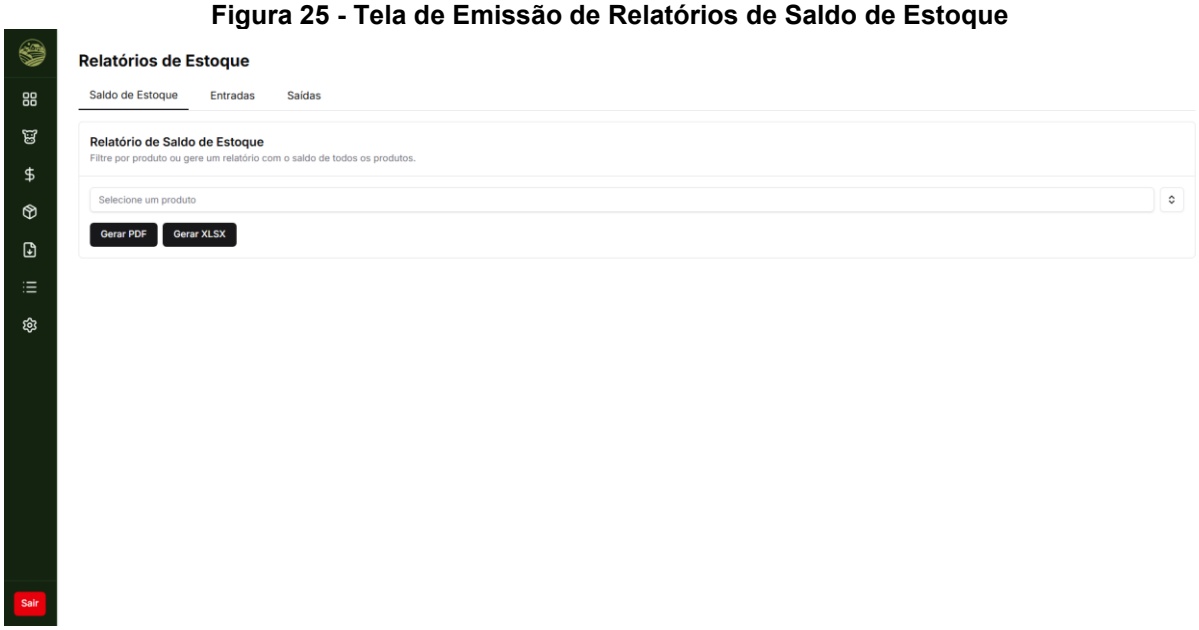


Fonte: Autoria própria (2025).

Para auxiliar no controle de estoque, garantindo a previsibilidade, é possível realizar a emissão de Relatórios de Estoque, que são divididos em três abas, uma para cada objetivo:

- Saldo de Estoque: Apresentada na Figura 25, essa aba permite gerar o relatório com o saldo atual de todos os itens cadastrados ou, opcionalmente, apenas de um item específico, selecionado por meio do filtro *Produto*.

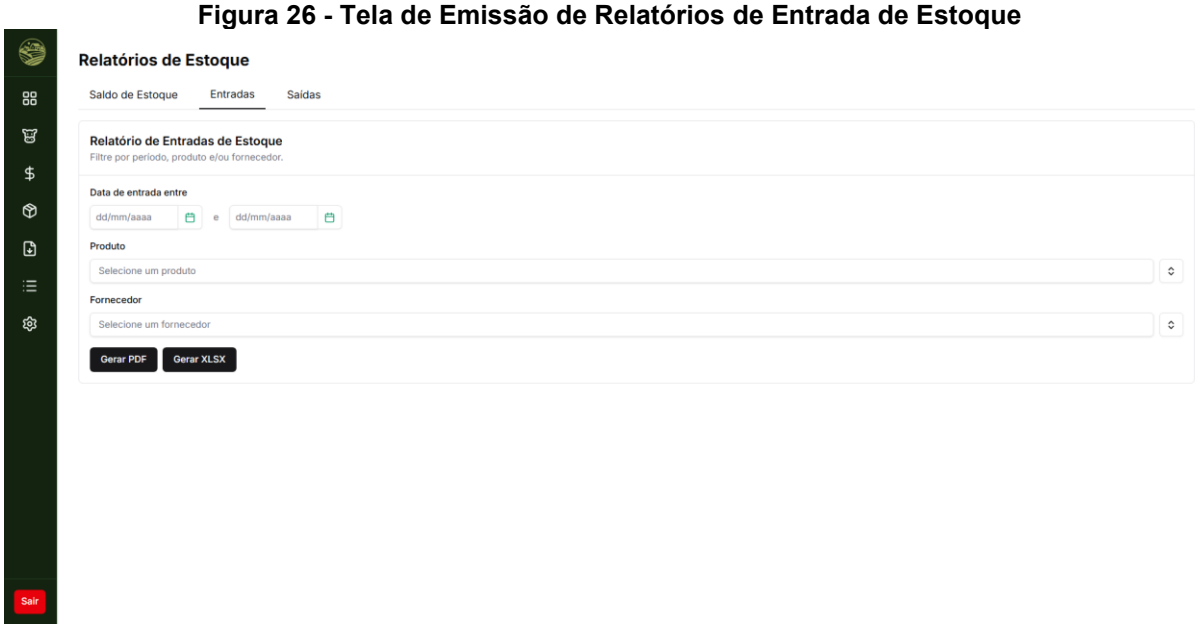
Figura 25 - Tela de Emissão de Relatórios de Saldo de Estoque



Fonte: Autoria própria (2025).

- Entradas: Conforme ilustrado na Figura 26, essa aba possibilita gerar relatórios contendo informações sobre as entradas de estoque. Os filtros disponíveis incluem “Data de entrada entre”, “Produto” e “Fornecedor”. Caso nenhum filtro seja aplicado, o sistema exibe todas as entradas registradas.

Figura 26 - Tela de Emissão de Relatórios de Entrada de Estoque



Fonte: Autoria própria (2025).

- Saídas: Na Figura 27 é apresentada a tela de emissão do relatório referente às saídas de estoque. Os filtros disponíveis são “Data de saída entre”, “Produto” e “Cliente”, de forma semelhante ao relatório de Entradas.

Figura 27 - Tela de Emissão de Relatórios de Saída de Estoque

Fonte: Autoria própria (2025).

4.4 Implementação do sistema

O desenvolvimento da aplicação foi estruturado em duas camadas principais: o *back-end*, implementado utilizando a linguagem Java em conjunto com o *framework* Spring Boot, e o *front-end*, desenvolvido com a biblioteca React. Para o armazenamento e gerenciamento dos dados, foi utilizado o banco de dados PostgreSQL, devido à sua confiabilidade, desempenho e aderência a padrões *Structured Query Language* (SQL).

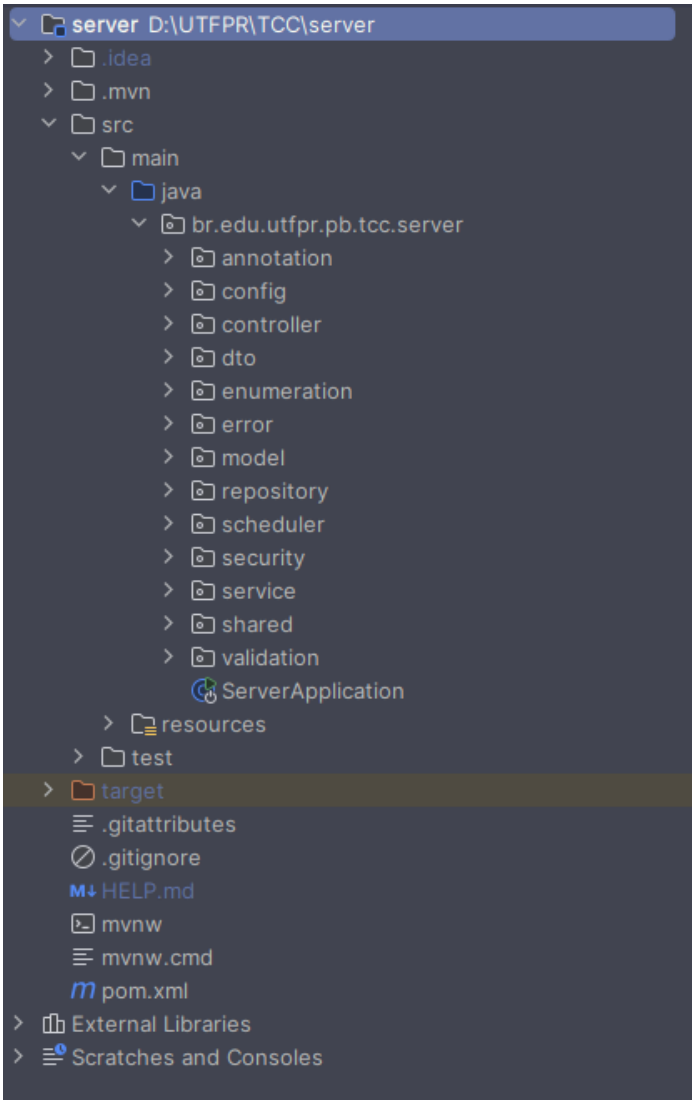
A escolha do Java e do Spring Boot se deve à confiabilidade e ampla utilização no mercado, além de oferecer uma arquitetura flexível baseada em *APIs REST*, o que facilita a criação de serviços escaláveis, seguros e de fácil manutenção. O Spring Boot também fornece uma série de recursos que reduzem a configuração manual, agilizando o desenvolvimento e padronizando a estrutura do projeto.

A aplicação segue um padrão arquitetural MVC que favorece separação de responsabilidades, garantindo clareza e modularidade no código. Na camada do *back-*

end, a estrutura é organizada em pacotes principais, conforme ilustrado na Figura 28, destacando-se:

- *controller*: responsável por receber e responder às requisições da aplicação, servindo como ponto de entrada da API.
- *model*: contém as classes que representam as entidades e seus atributos, correspondentes às tabelas do banco de dados.
- *repository*: reúne as interfaces responsáveis pela comunicação direta com o banco de dados, utilizando *Spring Data JPA*.
- *service*: implementa as regras de negócio, intermediando o fluxo entre *controller* e *repository*.

Figura 28 - Estrutura de pastas do *back-end*



Fonte: Autoria própria (2025).

No *front-end*, o React foi escolhido por sua abordagem baseada em componentes reutilizáveis, o que favorece a organização e manutenção da interface. Sua capacidade de atualização eficiente do DOM virtual, aliada ao uso de *hooks* e gerenciamento de estado, permite a criação de interfaces dinâmicas, responsivas e de alto desempenho, garantindo uma boa experiência ao usuário.

Essa organização favorece a manutenção do código, facilita futuras expansões do sistema e contribui para uma arquitetura limpa e bem estruturada, alinhada às boas práticas de desenvolvimento.

Para as consultas de dados do sistema, optou-se por uma abordagem híbrida que utiliza primariamente os recursos oferecidos pelo Spring Data JPA, equilibrando produtividade e a necessidade de filtros complexos. Essa estratégia se baseia em três

pilares principais: a derivação de consultas (*query derivation*) a partir de nomes de métodos, o uso de *JpaSpecificationExecutor* para consultas dinâmicas, e o uso pontual de consultas *Java Persistence Query Languag*e (JPQL) personalizadas para filtros mais específicos.

A principal técnica empregada é a derivação de consulta, onde o *Spring Data JPA* automaticamente gera as consultas *SQL* a partir da assinatura dos métodos definidos nas interfaces dos repositórios. Métodos como *findByStatusAndDueDateBetween* (no *AccountsPayableInstallmentRepository*) ou *findByCashAccountIdAndType* (no *CashTransactionRepository*) exemplificam essa abordagem. Isso permite um desenvolvimento rápido, legível e com baixa manutenção, pois a lógica da consulta está claramente expressa no nome do método, retornando listas ou páginas de entidades.

Para cenários que exigem uma filtragem mais dinâmica e programática, como nos cadastros de produtos e gestações, os repositórios estendem a interface *JpaSpecificationExecutor*. Essa abordagem permite a construção de consultas complexas baseadas em critérios (Criteria API), facilitando a aplicação de múltiplos filtros que podem ser combinados em tempo de execução, sem a necessidade de escrever manualmente cada variação de consulta no repositório.

Em situações onde a lógica de filtragem é mais específica e envolve múltiplas associações com parâmetros opcionais, optou-se por escrever consultas personalizadas usando JPQL por meio da anotação *@Query*. Diferente de uma abordagem focada em projeções para *Data Transfer Objects* (DTO), no entanto, essas consultas JPQL ainda retornam as entidades completas. Essa escolha simplifica a manutenção do código, pois não é preciso criar e manter *DTOs* de projeção para cada variação de consulta, ao mesmo tempo que resolve requisitos de filtro complexos.

A Listagem 1 apresenta um exemplo de consulta JPQL personalizada utilizada no *StockOutflowItemRepository*. Esta consulta demonstra como o JPQL foi empregado para buscar itens de saída de estoque, permitindo filtrar os resultados por um intervalo de datas, um produto específico (opcional) e um cliente/fornecedor específico (opcional), lidando com a lógica de parâmetros nulos diretamente na query.

Listagem 1 - Exemplo de consulta personalizada para filtros complexos

```
@Query("SELECT i FROM StockOutflowItem i LEFT JOIN i.stockOutflow.clientSupplier
cs WHERE " +
           "i.stockOutflow.outflowDate BETWEEN :startDate AND :endDate " +
```

```
            "AND (:productId IS NULL OR i.product.id = :productId) " +
            "AND (" +
            "    (:clientSupplierId IS NULL) OR " +
            "    (cs.id = :clientSupplierId)" +
            ")")
    List<StockOutflowItem> findOutflowsByDateRangeAndProductIdAndClientSupplierId(
            LocalDate startDate,
            LocalDate endDate,
            Long productId,
            Long clientSupplierId);
```

Fonte: Autoria própria (2025).

Para a geração dos dados apresentados no *dashboard* principal do sistema, foram implementadas consultas específicas, notavelmente na interface *CashAccountRepository*. De forma similar ao exemplo, foi utilizada a linguagem *JPQL* com projeção direta para objetos *DTOs*. Essa abordagem é empregada para otimizar a performance, buscando apenas os dados agregados necessários para a visualização, como o saldo total e as séries temporais para os gráficos.

Essas consultas são utilizadas na classe *DashboardServiceImpl*, que é responsável por reunir os dados de diferentes fontes em um único objeto, denominado *DashboardDto*. Esse objeto agrupa informações como o saldo total das contas (*CashBalanceDto*), um resumo de parcelas pendentes (*PendingInstallmentsSummaryDto*) e os dados diários para o gráfico de fluxo de caixa (*CashAccountDailySeriesDto*).

Para os dados que compõem os gráficos e resumos de parcelas, a consulta principal no serviço exige filtros obrigatórios de data inicial e data final. A Listagem 2 mostra a implementação da classe *DashboardServiceImpl*, que orquestra a busca e a consolidação desses dados.

Listagem 2 - Classe *DashboardServiceImpl*

```
@Service
public class DashboardServiceImpl implements IDashboardService {

    private final CashAccountRepository cashAccountRepository;
    private final AccountsPayableInstallmentRepository
accountsPayableInstallmentRepository;
    private final AccountsReceivableInstallmentRepository
accountsReceivableInstallmentRepository;

    public DashboardServiceImpl(CashAccountRepository cashAccountRepository,
                                AccountsPayableInstallmentRepository
accountsPayableInstallmentRepository,
                                AccountsReceivableInstallmentRepository
accountsReceivableInstallmentRepository) {
        this.cashAccountRepository = cashAccountRepository;
```

```
        this.accountsPayableInstallmentRepository =
accountsPayableInstallmentRepository;
        this.accountsReceivableInstallmentRepository =
accountsReceivableInstallmentRepository;
    }

    @Override
    public DashboardDto getDashboardData(LocalDate startDate, LocalDate endDate) {
        CashBalanceDto cashBalance = cashAccountRepository.findCashBalance();
        PendingInstallmentsSummaryDto pendingSummary =
getPendingInstallmentsSummary(startDate, endDate);
        List<CashAccountDailySeriesDto> dailySeries =
cashAccountRepository.findCashAccountDailySeries(startDate, endDate);

        return DashboardDto.builder()
                .cashBalance(cashBalance)
                .pendingInstallmentsSummary(pendingSummary)
                .cashAccountDailySeries(dailySeries)
                .build();
    }

    private PendingInstallmentsSummaryDto getPendingInstallmentsSummary(LocalDate
startDate, LocalDate endDate) {
        List<AccountsPayableInstallment> payableInstallments =
accountsPayableInstallmentRepository
                .findByStatusAndDueDateBetween(InstallmentStatus.PENDING,
startDate, endDate);

        List<AccountsReceivableInstallment> receivableInstallments =
accountsReceivableInstallmentRepository
                .findByStatusAndDueDateBetween(InstallmentStatus.PENDING,
startDate, endDate);

        BigDecimal payableTotal = payableInstallments.stream()
                .map(AccountsPayableInstallment::getValue)
                .reduce(BigDecimal.ZERO, BigDecimal::add);

        BigDecimal receivableTotal = receivableInstallments.stream()
                .map(AccountsReceivableInstallment::getValue)
                .reduce(BigDecimal.ZERO, BigDecimal::add);

        BigDecimal balance = receivableTotal.subtract(payableTotal);

        return new PendingInstallmentsSummaryDto(payableTotal, receivableTotal,
balance);
    }
}
```

Fonte: Autoria própria (2025).

No lado do cliente, o componente *DashboardPage* gerencia a exibição do *dashboard*. Ele utiliza *hooks* como o *useState* para controlar os filtros definidos pelo usuário (mês e ano) e para armazenar os dados recebidos da API.

A lógica de busca de dados é encapsulada no *hook useEffect*, acionado sempre que os filtros de mês ou ano são alterados. O *useEffect* realiza uma requisição

assíncrona ao *endpoint* */dashboard* da API, enviando os parâmetros *startDate* e *endDate*. O *IDashboardDto* retornado é então salvo no estado do componente.

A partir desse estado (*data*), as informações são distribuídas aos componentes de visualização da biblioteca Recharts. Para otimizar a performance, o *hook useMemo* é utilizado para processar e transformar dados brutos (como *data.cashFlowDailySeriesByAccount*) no formato exato exigido pelo gráfico de linhas (*LineChart*). Outros dados, como *data.productBalances* e *data.pendingInstallments*, são passados diretamente aos componentes de gráfico de barras (*BarChart*) e aos cartões de resumo. Essa abordagem reativa e modular facilita a manutenção.

A Listagem 3 demonstra os *hooks useEffect* (para a requisição dos dados) e *useMemo* (para o processamento dos dados do gráfico).

Listagem 3 - Requisição dos dados do dashboard

```
export function DashboardPage() {
  const [data, setData] = useState<IDashboardDto | null>(null);
  const [birthsMonthCount, setBirthsMonthCount] = useState<number | null>(null);

  const now = new Date();
  const [selectedYear, setSelectedYear] = useState(now.getFullYear());
  const [selectedMonth, setSelectedMonth] = useState(now.getMonth() + 1);

  const getMonthBounds = (y: number, m: number) => {
    const last = new Date(y, m, 0).getDate();
    return {
      startIso: `${y}-${String(m).padStart(2, "0")}-01`,
      endIso: `${y}-${String(m).padStart(2, "0")}-${String(last).padStart(2,
"0")}`,
    };
  };

  useEffect(() => {
    const { startIso, endIso } = getMonthBounds(selectedYear, selectedMonth);

    api.get<IDashboardDto>("/dashboard", {
      params: { startDate: startIso, endDate: endIso },
    })
    .then(res => setData(res.data))
    .catch(() => setData(null));
  }, [selectedYear, selectedMonth]);

  useEffect(() => {
    const { startIso, endIso } = getMonthBounds(selectedYear, selectedMonth);

    api.get("/gestation/filter", {
      params: { expectedBirthDateFrom: startIso, expectedBirthDateTo: endIso },
    })
    .then(res => setBirthsMonthCount(res.data?.totalElements ?? 0))
    .catch(() => setBirthsMonthCount(0));
```

```
  }, [selectedYear, selectedMonth]);

  const chartData = useMemo(() => {
    return data?.cashFlowDailySeriesByAccount?.[0]?.points?.map(p => ({
      date: p.date,
      amount: p.amount,
    })) ?? [];
  }, [data]);

  return null;
}
```

Fonte: Autoria própria (2025).

Para garantir o gerenciamento proativo de eventos financeiros e operacionais críticos, foi desenvolvida uma rotina automatizada de envio de lembretes. Esta funcionalidade visa alertar o usuário, via WhatsApp, sobre vencimentos de contas a pagar e sobre datas de partos previstos de gestações, assegurando que ações necessárias sejam tomadas no tempo correto.

A execução periódica dessa rotina foi implementada utilizando o recurso de agendamento de tarefas do Spring Framework. Por meio da anotação *@Scheduled*, configurou-se a execução dos métodos responsáveis pelas verificações. O método principal, *runReminders()*, é executado todos os dias às 9:30 da manhã, configurado pela expressão cron "0 30 9 * * *". Para garantir a consistência da execução, a anotação especifica o fuso horário de São Paulo (*zone = "America/Sao_Paulo"*), assegurando que o processo seja executado de forma automática no horário correto, sem necessidade de intervenção manual.

A Listagem 4 apresenta a classe *ReminderScheduler*, com foco na implementação do agendamento.

Listagem 4 - Classe de agendamento da rotina de Notificações

```
@Component
public class ReminderScheduler {
    @Scheduled(cron = "0 30 9 * * *", zone = "America/Sao_Paulo")
    @Transactional
    public void runReminders() {
        logger.info("Iniciando verificação de lembretes...");
        ReminderConfig config = configService.getGlobalConfig();
    }
}
```

Fonte: Autoria própria (2025).

A lógica que define se um lembrete deve ou não ser enviado, contida no método *runReminders()*, é baseada nas configurações globais do sistema. Inicialmente, o

sistema verifica se qualquer lembrete está habilitado e se um número de *WhatsApp* foi configurado. Em caso afirmativo, a rotina se divide em duas verificações principais: *checkAccountsPayableReminders* para contas a pagar e *checkGestationReminders* para gestações.

A verificação de contas a pagar busca por parcelas pendentes (*InstallmentStatus.PENDENTE*) em três cenários, com base nos dias definidos pelo usuário: próximo do vencimento, vencendo no dia, e atrasadas. A verificação de gestação realiza uma busca similar por partos previstos próximo da data e no dia. Para garantir que não haja comunicações excessivas e repetitivas, o sistema utiliza um *ReminderLog* para checar, por meio do método *wasReminderSentToday()*, se um alerta para aquela entidade específica já foi enviado no dia corrente. Caso um envio seja realizado, ele é registrado no log pelo método *logReminder()*. A Listagem 5 apresenta a implementação da lógica descrita.

Listagem 5 - Classe de lógica da rotina de Lembretes

```
@Component
public class ReminderScheduler {

    private boolean wasReminderSentToday(Long entityId, String entityType, String
reminderType) {
        return
reminderLogRepository.findByEntityIdAndEntityTypeAndLogDateAndReminderType(
                entityId,
                entityType,
                LocalDate.now(),
                reminderType
        ).isPresent();
    }

    private void logReminder(Long entityId, String entityType, String
reminderType) {
        ReminderLog log = ReminderLog.builder()
                .entityId(entityId)
                .entityType(entityType)
                .reminderType(reminderType)
                .logDate(LocalDate.now())
                .build();
        reminderLogRepository.save(log);
    }

    @Transactional
    public void runReminders() {
        logger.info("Starting reminder check...");
        ReminderConfig config = configService.getGlobalConfig();

        if (config.isAnyReminderEnabled() && config.getWhatsapp() != null &&
!config.getWhatsapp().isEmpty()) {
```

```
            if (config.isEnableAccountsPayableReminder()) {
                checkAccountsPayableReminders(config);
            }

            if (config.isEnableGestationReminder()) {
                checkGestationReminders(config);
            }
        } else {
            logger.info("Lembretes desativados ou número de WhatsApp ausente.");
        }
    }

    private void checkAccountsPayableReminders(ReminderConfig config) {
        String whatsappNumber = config.getWhatsapp();
        LocalDate today = LocalDate.now();

        int daysBefore = config.getAccountsPayableDaysBefore() != null ?
config.getAccountsPayableDaysBefore() : 3;
        LocalDate targetDueDate = today.plusDays(daysBefore);

        List<AccountsPayableInstallment> upcomingInstallments =
apInstallmentRepository.findByStatusAndDueDateBetween(
                InstallmentStatus.PENDENTE, targetDueDate, targetDueDate);

        if (!upcomingInstallments.isEmpty()) {
            for (AccountsPayableInstallment installment : upcomingInstallments) {

                if (wasReminderSentToday(installment.getId(),
ENTITY_TYPE_AP_INSTALLMENT, REMINDER_TYPE_UPCOMING)) {
                    logger.warn("Lembrete de vencimento futuro para parcela de
Contas a Pagar ID {} ignorado: já enviado hoje.");
                    continue;
                }
                whatsAppService.sendMessage(whatsappNumber, message);
                logReminder(installment.getId(), ENTITY_TYPE_AP_INSTALLMENT,
REMINDER_TYPE_UPCOMING);
            }
        }

    }

}
```

Fonte: Autoria própria (2025).

Para efetuar o envio de mensagens de lembrete via WhatsApp, foi utilizada a Whapi.cloud, uma plataforma que fornece uma API RESTful para a comunicação com a infraestrutura do WhatsApp. Diferente de bibliotecas com *Software Development Kit* (SDK) nativos, essa abordagem utiliza requisições HTTP diretas, o que oferece grande flexibilidade.

A implementação da lógica de envio está encapsulada na classe *WhatsAppServiceImpl*. Foi utilizado o *WebClient* do *Spring WebFlux* para construir e executar as requisições *POST* de forma assíncrona. Conforme as boas práticas de segurança, o token de autenticação e a URL da API não são fixos no código-fonte; eles são injetados a partir do arquivo de propriedades (*application.properties*).

A Listagem 6 exibe o código responsável por montar a requisição HTTP, adicionar o token de autenticação no cabeçalho e enviar a mensagem formatada para a API.

Listagem 6 - Classe responsável pelo envio de mensagens via WhatsApp

```
@Service
public class WhatsAppServiceImpl implements IWhatsAppService {

    private final WebClient webClient;
    private final String API_TOKEN;
    private static final Logger logger =
LoggerFactory.getLogger(WhatsAppServiceImpl.class);

    public WhatsAppServiceImpl(WebClient.Builder webClientBuilder,
                               @Value("${whapi.url}") String apiUrl,
                               @Value("${whapi.token}") String apiToken) {
        this.webClient = webClientBuilder.baseUrl(apiUrl).build();
        this.API_TOKEN = apiToken;
    }

    @Override
    public void sendMessage(String to, String message) {
        String cleanTo = to.replaceAll("[^0-9]", "");

        String formattedTo = "55" + cleanTo + "@s.whatsapp.net";

        WhatsAppRequest requestBody = WhatsAppRequest.builder()
                .to(formattedTo)
                .body(message)
                .build();

        logger.info("Enviando mensagem via WhatsApp para " + formattedTo);

        webClient.post()
                .uri("/messages/text")
                .header(HttpHeaders.CONTENT_TYPE,
MediaType.APPLICATION_JSON_VALUE)
                .header(HttpHeaders.AUTHORIZATION, "Bearer " + API_TOKEN)
                .body(BodyInserters.fromValue(requestBody))
                .retrieve()
                .bodyToMono(String.class)
                .doOnSuccess(response -> logger.info("WhatsApp: Mensagem
enviada com sucesso."))
                .doOnError(error -> logger.error("WhatsApp: Erro ao enviar
mensagem: {}", error.getMessage()))
                .subscribe();
```

```
    }

    @Data
    @Builder
    private static class WhatsAppRequest {
        private String to;
        private String body;
    }
}
```

Fonte: Autoria própria (2025).

Uma regra de negócio central no módulo de pecuária é o cálculo automático da data prevista de nascimento das gestações. Este cálculo baseia-se n período de gestação bovina, que normalmente dura em torno de nove meses, podendo apresentar pequenas variações conforme características individuais e a raça do animal. Em geral, esse intervalo situa-se entre aproximadamente 279 e 287 dias (Agroline, 2025). No sistema, o autor assumiu a média dessas duas datas, resultando em 283 dias.

Para garantir a integridade e a consistência dessa regra, a lógica principal foi implementada diretamente na camada de persistência, no *back-end*. Na entidade *Gestation*, foi utilizado um método de *callback* do ciclo de vida da JPA. Como demonstrado na Listagem 7, o método *onSave()*, anotado com *@PrePersist* e *@PreUpdate*, é invocado automaticamente antes de qualquer operação de inserção ou atualização no banco de dados. Este método chama a função *calculateExpectedBirthDate()*, que aplica a regra de negócio somando 283 dias à data de fertilização (*fertilizationDate*) informada, assegurando que o campo *expectedBirthDate* esteja sempre correto na base de dados.

Listagem 7 - Cálculo automático da data prevista de nascimento

```
public class Gestation {

    @NotNull(message = "Data da fertilização é obrigatória.")
    private LocalDate fertilizationDate;

    @NotNull(message = "Data prevista do nascimento é obrigatória.")
    private LocalDate expectedBirthDate;

    private LocalDate actualBirthDate;

    @PrePersist
    @PreUpdate
```

```
    public void onSave() {
        calculateExpectedBirthDate();
    }

    public void calculateExpectedBirthDate() {
        if (this.fertilizationDate != null) {
            this.expectedBirthDate = this.fertilizationDate.plusDays(283);
        }
    }
}
```

Fonte: Autoria própria (2025).

Para complementar esta abordagem de *back-end* e melhorar a experiência do usuário, a mesma lógica de cálculo foi replicada no *front-end*. No componente de formulário de gestação (*GestationEditDialog*), um *hook useEffect* do React monitora o campo da data de fertilização. Quando este campo é alterado, o *hook* recalcula a data prevista de nascimento em tempo real (também somando 283 dias) e atualiza o campo correspondente no formulário, fornecendo *feedback* visual imediato ao usuário antes mesmo que os dados sejam salvos.

4.5 Testes

A validação do sistema foi realizada por meio de um processo de testes em múltiplas camadas, abrangendo tanto o *back-end* (API) quanto o *front-end* (interface do usuário), para assegurar a corretude das regras de negócio e a fluidez da experiência de uso.

Inicialmente, foi utilizada a ferramenta Postman para a realização de testes de API. Esta abordagem permitiu validar isoladamente a lógica de negócio e as regras de persistência de dados em todos os *endpoints* RESTful do servidor, antes mesmo da integração com a interface. Foram testadas rigorosamente as operações de CRUD para todas as entidades principais, garantindo que os dados eram processados, calculados e validados corretamente pela API. Caso necessário, os testes via Postman podem ser realizados novamente em ambiente teste, para assegurar a correta configuração do *back-end*.

Em paralelo, foram realizados testes manuais *end-to-end* (E2E) diretamente na interface do usuário. Estes testes envolveram simulações de cenários de uso real, desde o cadastro de contas de caixa e transações até a geração de relatórios e a visualização do dashboard. Nesses cenários, verificou-se a precisão dos cálculos de

fluxo de caixa, a correta categorização das transações e a adequada apresentação das informações gráficas.

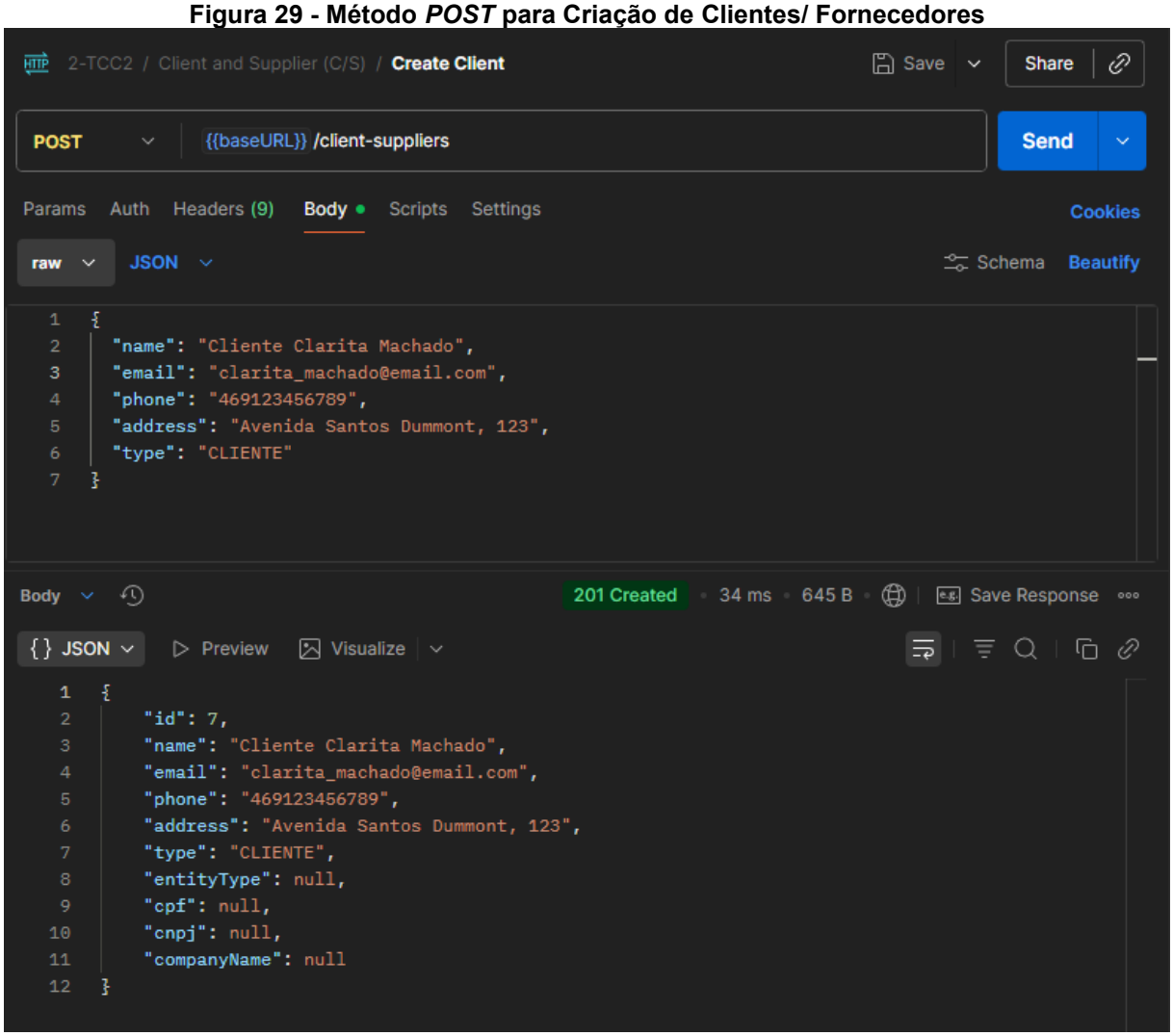
Além disso, foram feitos testes específicos para a funcionalidade de lembretes via WhatsApp. Para isso, datas do sistema e das entidades foram manipuladas para simular vencimentos futuros, no dia e atrasados, assegurando que os alertas agendados fossem disparados e enviados corretamente. Esse processo iterativo de testes, combinando a validação de API com a validação manual da interface, foi fundamental para aprimorar a confiabilidade e a qualidade geral do sistema, garantindo que todos os requisitos funcionais foram atendidos.

4.5.1 Testes de API

Como descrito, a ferramenta Postman foi central para validar as requisições e validações do *back-end*. A seguir, são apresentados exemplos das principais operações (CRUD).

- Testes de Criação (*POST*): A Figura 29 demonstra a criação de um novo Cliente (*endpoint POST /client-suppliers*). O corpo da requisição (*body*) é enviado em formato JavaScript Object Notation (JSON) com os dados do cliente, e a API retorna o status “201 *Created*” e o objeto criado, confirmando a persistência.

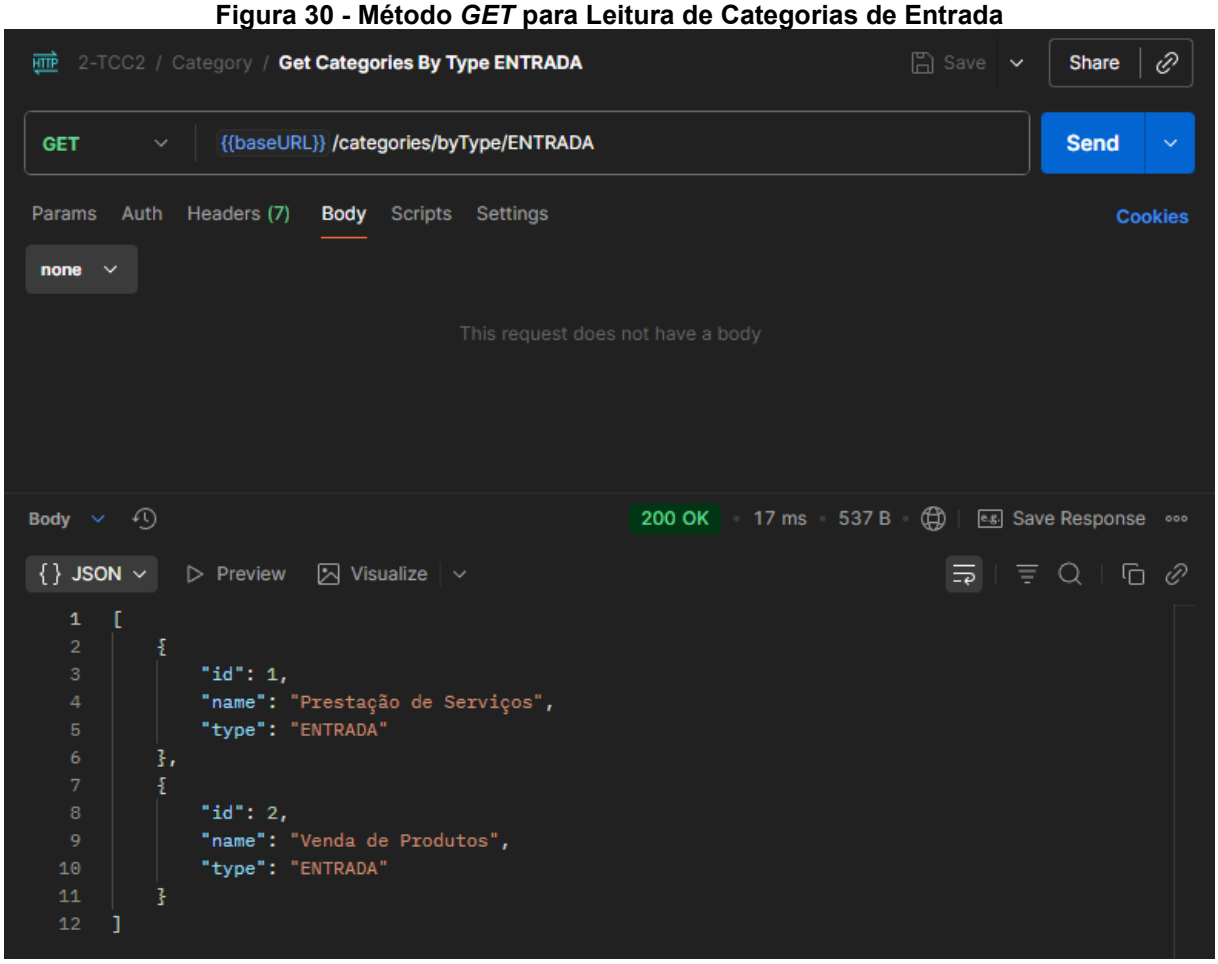
Figura 29 - Método *POST* para Criação de Clientes/ Fornecedores



Fonte: Autoria própria (2025).

- Testes de Leitura (*GET*): A Figura 30 ilustra uma requisição *GET* para consultar categorias por tipo (*GET /categories/byType/ENTRADA*). A API retorna o status “200 OK” e uma lista (*array*) de objetos JSON que correspondem ao critério solicitado.

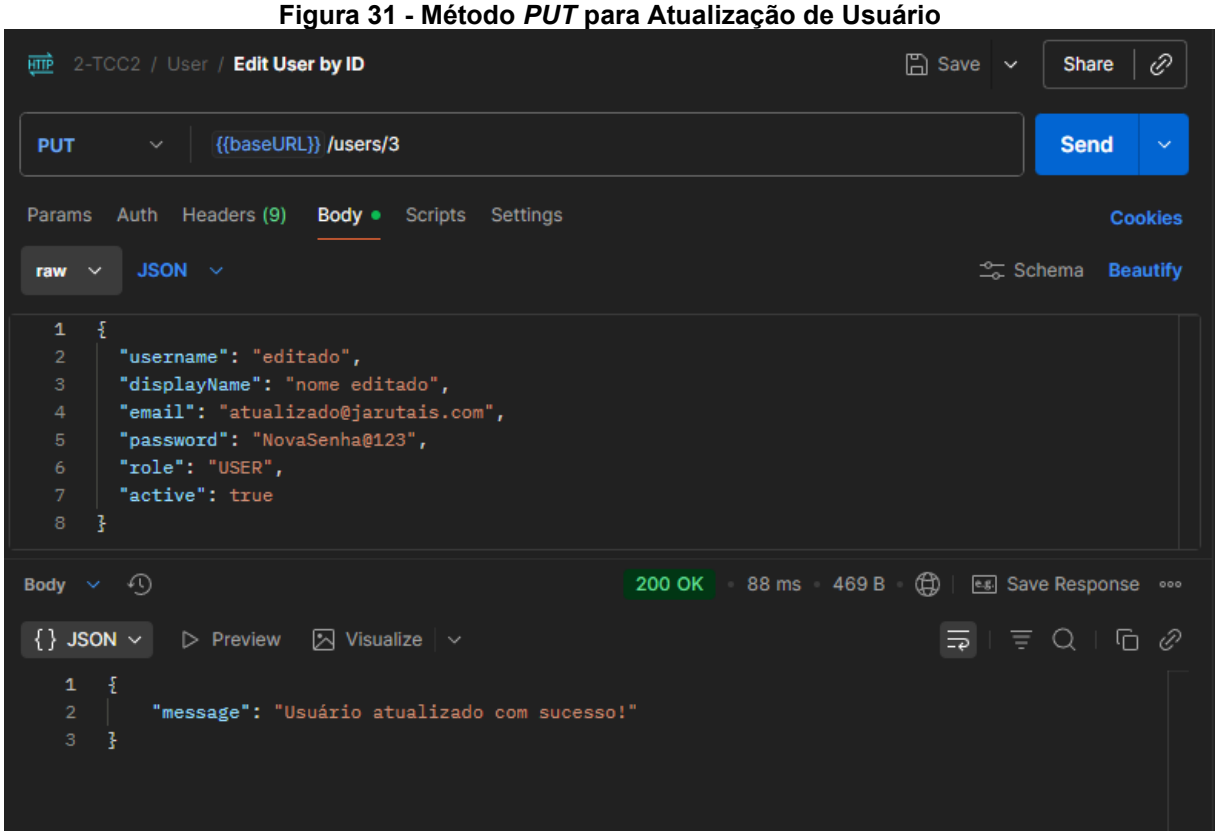
Figura 30 - Método *GET* para Leitura de Categorias de Entrada



Fonte: Autoria própria (2025).

- Testes de Atualização (*PUT*): A Figura 31 apresenta a atualização de um usuário existente, passando o *ID* do usuário (*PUT /users/{id}*). O corpo da requisição contém os dados modificados. A API processa a alteração e retorna o status “200 OK” e uma mensagem confirmando a alteração.

Figura 31 - Método *PUT* para Atualização de Usuário



Fonte: Autoria própria (2025).

- Testes de Exclusão (*DELETE*): A Figura 32 exibe a exclusão que não foi bem sucedida de um método de pagamento (*DELETE /charge-methods/1*). Esta operação é um exemplo chave onde o *back-end* aplica uma validação em duas camadas.

  A primeira camada é a de permissão (Autorização). A requisição exige que o usuário autenticado possua a *role* de “ADMIN” para ser executada.

  - Cenário de Teste 1 (Usuário Comum): Um *token* de usuário com *role* USER é enviado na requisição. O servidor recusa a operação e retorna o status “*403 Forbidden*”, antes mesmo de verificar o banco de dados.
  - Cenário de Teste 2 (Administrador): Um token de usuário com role ADMIN é enviado. A API passa para a segunda camada de validação.

  A segunda camada é a de integridade referencial dos dados (Regra de Negócio). Antes de excluir, o serviço de *back-end* verifica se o registro (nesse exemplo, o "Método de Pagamento") possui vínculos com outras tabelas.

  - Cenário de Teste 2.a (Registro Vinculado): O *ChargeMethod* de *ID* 1 já foi utilizado em um lançamento de *AccountsPayable* (Contas a Pagar)

ou *StockOutflow* (Saída de Estoque). O banco de dados (por meio de uma *Foreign Key constraint*) ou a lógica de serviço do *back-end* impede a exclusão. O teste valida que a API retorna o status de erro "*409 Conflict*", informando que o registro está em uso e não pode ser removido. A Figura 32 ilustra esse conflito.

- Cenário de Teste 2.b (Registro Livre): O *ChargeMethod* de ID 1 não possui nenhum vínculo (nenhuma conta a pagar ou venda o utiliza). Sendo o usuário um ADMIN e o registro estando livre, a API processa a exclusão com sucesso e retorna o status *"204 No Content"*, confirmando a remoção.

Figura 32 - Método *DELETE* para Exclusão de Meio de Pagamento

Fonte: Autoria própria (2025).

Este processo de teste duplo garante não apenas a segurança de acesso, mas também a consistência e a integridade da base de dados, evitando registros "órfãos".

4.5.2 Testes de Envio de Lembretes via WhatsApp

Adicionalmente, foram feitos testes específicos e manuais para a funcionalidade de lembretes via WhatsApp. Esta etapa foi crucial, pois envolvia a integração com a API externa Whapi.cloud, cuja comunicação é gerenciada pela classe *WhatsAppServiceImpl*.

O processo de teste seguiu os seguintes passos:

- Configuração: Primeiramente, foi necessário configurar um número de telefone válido e habilitar os lembretes na tela de "Configuração de Lembretes" do sistema (que utiliza o *endpoint PUT* /*reminder-config/global*).
- Criação de Massa de Dados: As datas de vencimento de parcelas e as datas previstas de parto foram manipuladas (seja via API, *front-end*, ou diretamente na base de dados) para forçar a ativação das regras de negócio no *ReminderScheduler*, simulando os diferentes cenários de alertas (próximos do vencimento, no dia e atrasados).
- Execução e Verificação (Dupla): Para facilitar a validação rápida, a configuração do *scheduler* (definido no *ReminderScheduler*) foi ajustada especificamente no ambiente de testes. Em vez de rodar apenas uma vez ao dia (como em produção), a tarefa foi programada, usando uma expressão cron, para ser executada a cada 30 segundos. Desta forma, após a criação da massa de dados, bastou aguardar o próximo ciclo para a verificação ser disparada.

O sucesso dos testes foi validado em duas frentes:

- Validação do Efeito (Figura 33): Por meio da observação direta do recebimento das mensagens formatadas corretamente no aparelho de destino.

Figura 33 - Mensagens Recebidas no WhatsApp Configurado

- Validação da Causa (Figura 34): A partir da análise do log de envio no servidor. Este log foi crucial, pois confirmou que o *ReminderScheduler* foi executado no horário correto, identificou as parcelas pendentes e acionou o *WhatsAppServiceImpl*, registrando a tentativa de comunicação com a API externa e evitando que seja enviado mais de uma vez por dia.

Figura 34 - Tabela *Reminder_Log* para Registro das Notificaçõed Enviadas

	id [PK] bigint	entity_id bigint	entity_type character varying (255)	log_date date	reminder_type character varying (255)
1	64	71	ACCOUNTS_PAYABLE_INSTALLMENT	2025-11-11	UPCOMING
2	65	72	ACCOUNTS_PAYABLE_INSTALLMENT	2025-11-11	ON_DUE_DATE
3	66	73	ACCOUNTS_PAYABLE_INSTALLMENT	2025-11-11	OVERDUE
4	67	4	GESTATION	2025-11-11	GESTATION_DUE
5	68	3	GESTATION	2025-11-11	ON_BIRTH_DATE
6	69	5	GESTATION	2025-11-11	ON_BIRTH_DATE

Fonte: Autoria própria (2025).

5 CONCLUSÃO

Neste trabalho foi desenvolvido um sistema *web* de gestão para propriedades rurais, focado em atender às necessidades de controle financeiro, de estoque e de rebanho. A concepção desse sistema surgiu a partir da observação de uma necessidade familiar, onde o controle de contas, de insumos e de eventos pecuários (como datas de fertilização e partos) era realizado de forma manual em cadernos e calendários físicos. O objetivo principal foi criar um ambiente unificado e digital para centralizar essas informações, aprimorando o controle gerencial e facilitando a tomada de decisões baseada em dados.

A principal característica do sistema é a capacidade de integrar os diferentes módulos da propriedade. O usuário pode realizar a gestão de contas a pagar e receber, incluindo o cadastro de categorias para um controle detalhado das entradas e saídas de caixa e a efetiva gestão financeira da propriedade. Além disso, pode controlar o fluxo de caixa e, ao mesmo tempo, gerenciar o controle de estoque de rações e insumos. O sistema permite não apenas o controle de saldo, mas também o registro detalhado de uso e de eventuais vendas desses itens, obtendo assim uma visão holística da operação.

O sistema também oferece um módulo de pecuária, que se destaca pela capacidade de gerenciar gestações, vacinações e o cadastro individual do rebanho. Uma regra de negócio importante é o cálculo automático da data prevista de parto com base na data de fertilização. Este módulo é desenhado para visualizar e prever a data da natalidade dos bovinos, permitindo a geração de relatórios detalhados que incluam informações cruciais como a data de concepção ou de inseminação (em casos de reprodução artificial). Aliada a isso, a funcionalidade de lembretes automáticos via WhatsApp garante que o produtor seja notificado sobre compromissos financeiros, emitindo mensagens de alerta sobre vencimentos de contas a pagar, e datas de manejo importantes, como partos previstos, evitando perdas e otimizando o tempo.

O painel de controle, ou dashboard, oferece informações visuais e rápidas sobre a saúde da propriedade, incluindo o saldo total das contas, o fluxo de caixa diário, partos previstos para o mês e os produtos com maior saldo em estoque. Os recursos de filtragem e os relatórios possibilitam uma análise mais profunda e personalizada da operação. Isso inclui a possibilidade de classificar itens do estoque

e emitir relatórios detalhados de entradas, saídas e saldo, complementando a visão gerencial.

Em resumo, este sistema fornece uma plataforma centralizada para a gestão rural, ajudando pequenos e médios produtores a alcançarem seus objetivos e a tomar decisões informadas, substituindo os controles manuais suscetíveis a erros. Este trabalho pode ser útil para propriedades que desejam modernizar sua administração e desenvolver uma maior consciência sobre suas finanças e operações.

A escolha das tecnologias, como Spring Boot com Java para o *back-end* e React com TypeScript para o *front-end*, foi motivada pela familiaridade do autor com o ecossistema Java e seu *framework* principal. Durante o desenvolvimento, a maior dificuldade enfrentada foi a implementação dos gráficos do *dashboard*, por ser necessário lidar com a agregação de dados de diferentes fontes (contas a pagar, a receber, saldos de estoque) em formatos específicos. Para solucionar isso, os dados foram processados e montados no lado do *back-end*, utilizando projeções JPQL para DTOs, permitindo que o *front-end* apenas consumisse o objeto retornado e desenhasse o gráfico.

Para versões futuras do sistema, considera-se o desenvolvimento de um aplicativo móvel (nativo ou híbrido). Tal funcionalidade agregaria um valor significativo, pois permitiria o registro de informações diretamente no campo (como a aplicação de uma vacina ou a saída de um item do estoque), mesmo em locais sem acesso imediato à internet, sincronizando os dados posteriormente.

REFERÊNCIAS

AGROLINE. **Período de gestação da vaca: entenda o ciclo completo**. Disponível em: https://blog.agroline.com.br/vaca-periodo-de-gestacao/. Acesso em: 5 nov. 2025.

EMBRAPA. SW Agro- **Estudo do mercado brasileiro de software para o agronegócio**. [s.n.], 2011. Disponível em: http://ainfo.cnptia.embrapa.br/digital/bitstream/item/59341/1/Livro-SWAgro-digital.pdf. Acesso em: 23 mar. 2024.

FAO, 2021. Organização das Nações Unidas para a Alimentação e a Agricultura. Disponível em https://www.fao.org/brasil/noticias/detail-events/pt/c/1397857/. Acesso em: 23 mar. 2024.

GOV.BR, 2024. Ministério da Agricultura e Pecuária. Disponível em https://www.gov.br/agricultura/pt-br/assuntos/noticias/mapa-destaca-acoes-voltadas-para-o-setor-agricola. Acesso em: 23 mar. 2024.

ORACLE. ***The Java Language Specification***. Disponível em: https://docs.oracle.com/javase/specs/. Acesso em: agosto de 2024.

VMware. **Spring boot *Documentation***. Disponível em: https://docs.spring.io/Spring-boot/index.html. Acesso em: 2 nov. 2024.

VMware. **Spring Data**. Disponível em: https://spring.io/projects/spring-data. Acesso em: 4 dez. 2024.

VMware. **Spring Data JPA**. Disponível em: https://docs.spring.io/spring-data/jpa/reference/#reference. Acesso em: 4 dez. 2024.

VMware. **17. Web MVC *framework***. Disponível em: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html. Acesso em: 4 dez. 2024.

COODESH, E. **O que é arquitetura MVC?** Disponível em: https://coodesh.com/blog/dicionario/o-que-e-arquitetura-mvc/. Acesso em: 5 dez. 2024.

GARCIA, Rodrigo. **Começando com Spring Security**. Medium, 2021. Disponível em: https://medium.com/cwi-software/come%C3%A7ando-com-spring-security-86a3caec8c40. Acesso em: 7 dez. 2024.

VMware. **Spring Security**. Disponível em: https://spring.io/projects/spring-security. Acesso em: 7 dez. 2024.

FIA. **Agronegócio: o que é, como funciona e setores**. Disponível em: https://fia.com.br/blog/agronegocio/. Acesso em: 11 dez. 2024.

TOTVS, E. **Agricultura de subsistência: o que é, como funciona e desafios.** Disponível em: https://www.totvs.com/blog/gestao-agricola/agricultura-de-subsistencia/. Acesso em: 12 dez. 2024.

DEITEL, H. M.; DEITEL, P. J. **Java: Como Programar**. 10. ed. São Paulo: Pearson, 2015.

ORACLE. ***What is the Java Platform?***. Disponível em: https://www.oracle.com/java/technologies/. Acesso em: 13 jan. 2025.

GOSLING, J.; JOY, B.; STEELE, G.; BRACHA, G. The Java Language Specification. 2. ed. Addison-Wesley, 2000.

NOGUEIRA, L. M.; SILVA, S. R.; OLIVEIRA, L. B. **Agricultura de precisão no Brasil: conjuntura atual, desafios e perspectivas**. Disponível em: https://www.researchgate.net/publication/346981141_Agricultura_de_Precisao_no_Brasil_conjuntura_atual_desafios_e_perspectivas . Acesso em: 13 jan. 2025.

PIERCE, F. J.; NOWAK, P. ***Precision farming: Current status and environmental impacts.*** **Advances in Agronomy**, v. 67, p. 1-85, 1999. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0065211308605131. Acesso em: 13 jan. 2025.

KNOB, M**. Aplicação de técnicas de agricultura de precisão em pequenas propriedades.** 2016. Dissertação (Mestrado em Engenharia Agrícola) – Universidade Federal de Santa Maria, Santa Maria, 2016. Disponível em:

https://repositorio.ufsm.br/bitstream/handle/1/7486/MARCELINO%20KNOB.pdf?sequence=1. Acesso em: 13 jan. 2025.

ALTIERI, M. A. **Agroecologia: bases científicas para uma agricultura sustentável**. São Paulo: Expressão Popular, 2002.

TEIXEIRA, L. A. et al. **Agroecologia e desenvolvimento rural sustentável: experiências no Brasil**. São Paulo: Universidade de São Paulo, 2010.

COSTA, A. L. et al. Agricultura de precisão: desafios e perspectivas para o uso de tecnologias no campo. **Revista Brasileira de Tecnologia Agropecuária**, v. 14, n. 3, p. 100-115, 2019.

LEMOS, M. et al. Sistemas de informação na gestão agrícola: contribuições para a sustentabilidade das propriedades rurais. **Revista de Gestão do Agronegócio**, v. 23, n. 1, p. 44-58, 2017.

META. ***Tutorial: Intro to React***. Disponível em: https://legacy.reactjs.org/tutorial/tutorial.html. Acesso em: 30 jan. 2025.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de Software: Uma abordagem profissional**. 9. ed. Porto Alegre/RS: AMGH Editora Ltda., 2021. 1035 p. ISBN 978-65-5804-011-8.

CHUNG, L.; LEITE, Julio Cesar Sampaio do Prado. ***On Non-Functional Requirements in Software Engineering. Conceptual Modeling: Foundations and Applications***, 4 jun. 2024. Disponível em: https://www.researchgate.net/publication/215697482_On_Non-Functional_Requirements_in_Software_Engineering. Acesso em: 01 nov. 2025.